Litt om CSS

Jan R Sandbakken

Litt om CSS

Jan R Sandbakken

Version 1.0 2026-02-12

Innholdsfortegnelse

Litt om CSS

Her skal vi se litt på grunnleggende CSS. CSS-standarden er stor og omhandler mange ting som det ikke er naturlig å ta for seg i en tidlig fase. Ting som ulike paneler, grids, bilder, tabeller mm, må vi la ligge i denne omgang. Det vil bli fokusert mest på tekstlige elementer.

Overordnet HTML

Helt grunnleggende HTML antas kjent. Bruk av overskrifter som `<h1>`, linker, ulike *mark-ups* for fet eller kursiv skrift og den slags, behandles ikke. Her fokuseres det på struktur med relevans for stiling.

Et HTML-dokument kan strukturelt sett se slik ut:

```
<html>
  ├──── <head> ⋯ metadata, CSS, title ⋯
  └──── <body> ──── hele synlige innholdet på siden
        ├──── <header> ⋯ toppbanner, logo, navigasjon ⋯
        ├──── <nav> ⋯ hovedmeny, lenker ⋯
        ├──── <main> ⋯ hovedinnholdet, unikt for denne siden
        │      ├──── <section> ⋯ logisk gruppering av innhold
        │      ├──── <article> ⋯ en artikkel, blogginnlegg, etc.
        │      └──── <div> ⋯ ekstra beholder for styling/layout
        ├──── <aside> ⋯ sidepanel, ekstra info, reklame ⋯
        └──── <footer> ⋯ bunntekst, copyright, kontaktinfo ⋯
```

Videre ser vi et HTML-eksempel, med syntaks og viktige elementer:

```
<!DOCTYPE html>
<html lang="no">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Litt om CSS</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <h1>Litt om CSS</h1>
        <p>Av Jan R Sandbakken</p>
    </header>

    <main>
        <section>
            <h2>Introduksjon</h2>
            <p>Dette er et eksempel på et avsnitt i HTML.</p>
            <pre>
Dette er preformatert tekst.
Beholder mellomrom og linjeskift.
            </pre>
        </section>

        <section>
            <h2>Eksempel på div</h2>
            <div class="boks">
                Dette er en div som ofte brukes for å style grupper av innhold.
            </div>
        </section>
    </main>

    <footer>
        <p>&copy; 2026 Jan R Sandbakken</p>
    </footer>
</body>
</html>
```

hvor referansen til CSS-filen som styler dokumentet, er gitt ved

```
<link rel="stylesheet" href="style.css">
```

Elementene som benyttes er

- `<header>` er overskriftsseksjon på toppen av dokumentet
- `<body>` er alt det synlige innholdet (inkl. header, footer, sidepanel, ...)
- `<main>` er hovedinnhold

- `<section>` er logiske grupper av innhold (f.eks. kapitler og emner)
- `<div>` er en generell beholder, typisk for avsnittspreget layout og styling
- `<p>` er vanlig avsnitt
- `<pre>` er preformatert tekst som beholder linjeskift og mellomrom
- `<footer>` er bunntekst

Andre viktige tekstlige elementer er bl.a:

- `<code>` – for formatering av kodeaktige ord
- `<span>` – generell beholder for mer ordbasert formatering

Hva er CSS?

CSS står for *Cascading Style Sheets*, som representerer:

- Cascading → Regler "kaskaderer", dvs. nye regler kan overstyre gamle basert på spesifisitet og rekkefølge.
- Style → Angir hvordan elementer ser ut: farge, størrelse, plassering, marg, osv.
- Sheets → Samles i en fil (f.eks. **style.css**) og kobles til HTML.

CSS kobles til HTML via:

1. Selektor – hva du vil style (f.eks. **p**, **.boks**, **header**)
2. Egenskap – hva du vil endre (f.eks. **color**, **font-size**, **margin**)
3. Verdi – hvordan du vil endre det (f.eks. **blue**, **1.2rem**, 1**0px**)

Eksempel:

```
/* Stiler alle <p>-avsnitt */
p {
    color: darkblue;
    line-height: 1.5;
}
```

Hovedprinsippene er at:

- CSS separerer struktur (HTML) fra utseende (CSS) → lettere vedlikehold.
- En regel består alltid av selektor + deklarasjonsblokk
- CSS arves nedover i dokumentet (f.eks. `color` fra `<body>` arves til `<p>`)

Konstruksjonen som bestemmer verdier, ser generelt slik ut:

```
selektor {
    egenskap: verdi;
    egenskap: verdi;
}
```

Når det gjelder arv, er det slik at noen egenskaper arves (som `color`) og andre ikke (som `padding`), og i tillegg er det et arvehierarki som gjelder I HTML/CSS (som f.eks `<body>` → `<div>` → `<p>`). Vi skal komme tilbake til dette, men de formelle spesifikasjonene er å finne på

- MDN CSS Reference
- CSS Specification (W3C)

Globale CCS-settinger

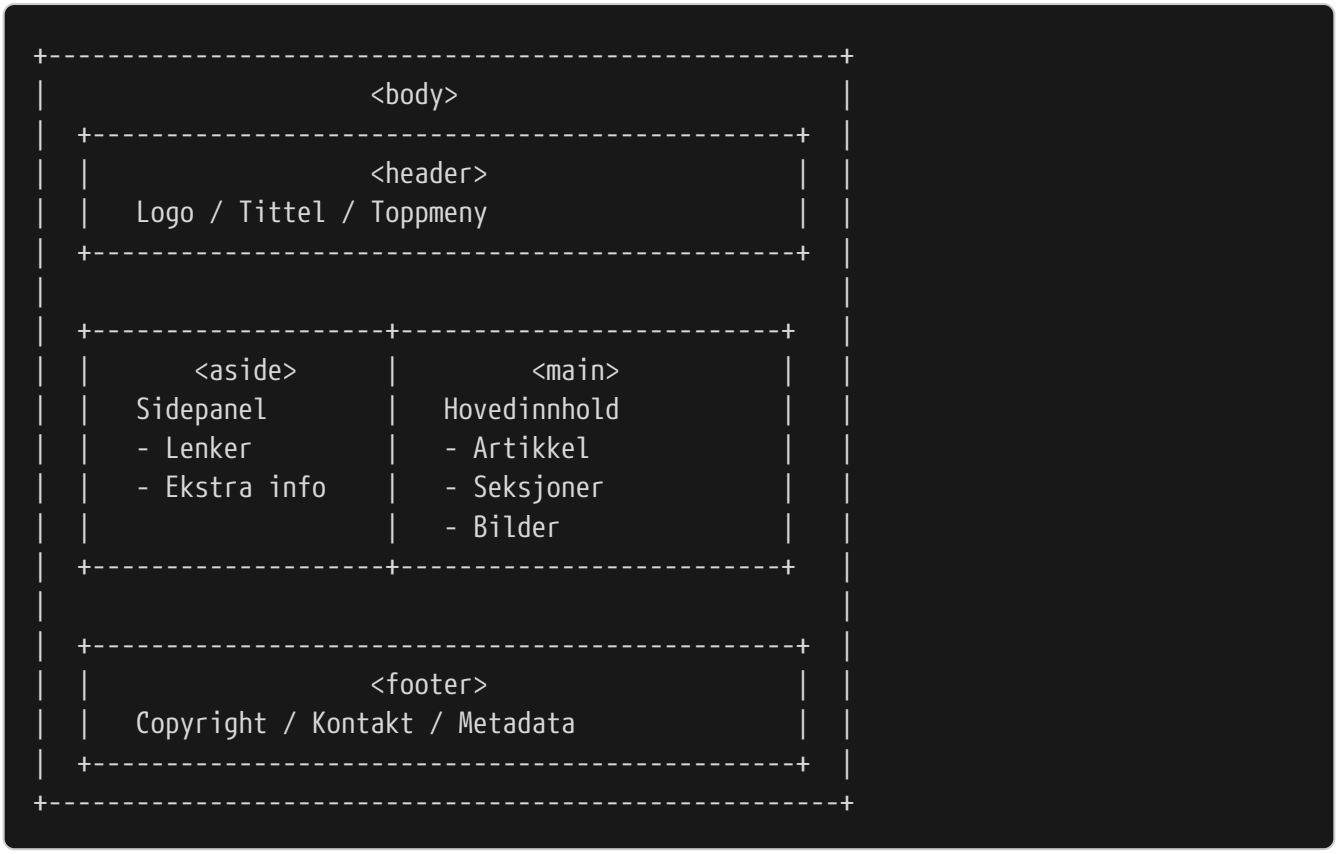
Her ser vi eksempel på noen globale settinger

```
html {
    font-size: 16px; /* gir grunnstørrelse for rem */
}

body {
    font-family: system-ui, sans-serif; /* lettlest standardfont */
    line-height: 1.6;                   /* behagelig linjeavstand */
    color: #222;                        /* tekstfarge */
    background-color: #f5f5f5;          /* bakgrunnsfarge */
}
```

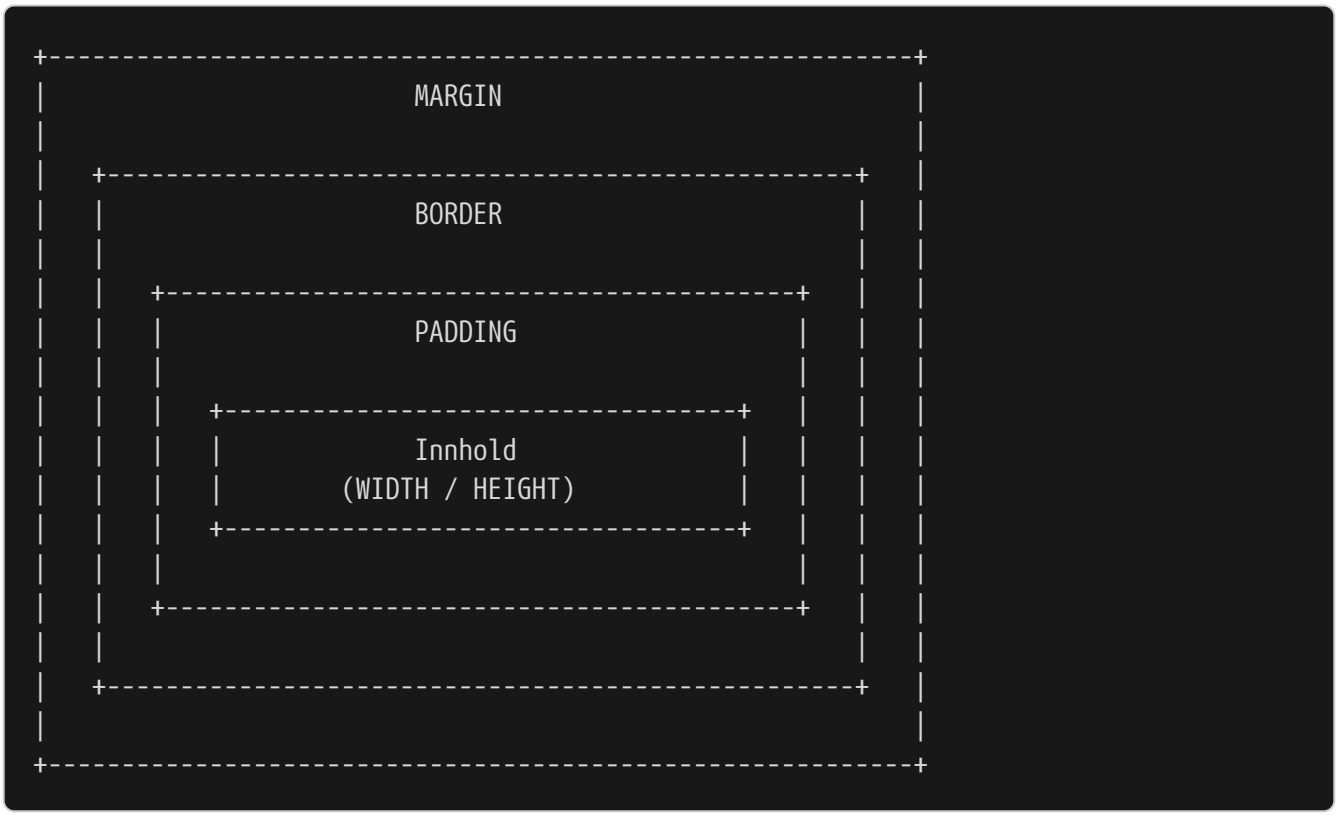
I HTML settes ting som bør være globalt, men sjelden knyttes til visuelt design. HTML-settinger er typisk språk og metadata, globale layout-verdier, scroll-behavior etc.

BODY representerer det synlig innholdet, laget som faktisk tegnes i vinduet. Her er det typisk å sette bakgrunnsfarge, font-family, fontfarge, marger og padding, layout-begrensning som max-width mm. BODY representerer hele sider, inklusive sidepanel, *footer* og *header* mm, og dette indikerer hva som er naturlig å sette her og ikke f.eks. i MAIN (som representerer hovedinnholdet, og som *kan* ha egen bakgrunnsfarge, marger etc. uten å påvirke topp og bunn og sidepanel fra BODY, hvis ønskelig). Følgende figur illustrerer situasjonen:



Boksmodellen

De fleste elementer har boksegenskaper, f.eks. overskrifter, **p**, **div**, **span**, **section**, **article**, **image** m.fl. Dvs. de kan ha marger, rammer og ulike former for luft rundt seg. Dette er konkret vist i figuren under.



Vi ser at størrelsen av innholdet kan spesifiseres ved **width** og **height**, og man kan angi luft

(**padding**) og ramme rundt dette (**border**). Rundt dette igjen, rundt selve boksen, kan man så også legge inn luft i form av **margin**.

Content box vs Border box

Vi har to typer bokser, *content* og *border box*. For en content box (som er default) kommer **padding** og **border** i tillegg i bestemmelse av endelig størrelse. For border-box er spesifikk størrelse også den faktisk størrelsen. (Husk at **margin** alltid gjelder utenfor boksen, uansett hvilken box-sizing man bruker.)

Slik settes de:

```
/* content-box: standard */
div.content-box {
  box-sizing: content-box;
  width: 300px;
  padding: 20px;
  border: 5px solid black;
}

/* border-box: praktisk for layout */
div.border-box {
  box-sizing: border-box;
  width: 300px;
  padding: 20px;
  border: 5px solid black;
}
```

Angivelse av størrelse

Måter å spesifisere størrelse på er: **px**, **em**, **rem**, **%**, **vw** og **vh**

De brukes på ikke bare på **margin**, **padding** osv, men også på **width**, **font-size** o.l.

px og **em** er de vanligste.

px

px angir størrelse i antall piksler. Den er ment å gi en absolutt, forutsigbar verdi. En boks med bredde 300px vil prøve å være nøyaktig 300 piksler bred uavhengig av skjermstørrelse.

Verdier kan settes direkte som her:

```
div {
  margin: 10px;
  padding: 20px;
}
```

For egenskaper som **margin** og **padding**, som kan ha verdier på flere sider av et innhold, vil rekkefølgen av tallene bestemme siden de angir. Rekkefølgen er iht. **top** → **right** → **bottom** → **left**, altså med klokka. Dvs. at

```
div {
  margin: 10px 20px 30px 40px;
}
```

angir størrelser etter

```
          10px
   40px             20px
          30px
```

Man har også definert betydningene

```
div {
    margin: 10px;                  /* alle */
    margin: 10px 20px;             /* T/B  R/L */
    margin: 10px 20px 30px;        /* T  R/L  B */
    margin: 10px 20px 30px 40px;   /* T  R  B  left */
}
```

Videre har man mulighet for å bruke `margin-top`, `margin-right`, `margin-bottom` og `margin-left`, og det tilsvarende også for `padding` og `border`. Men sistnevnt består egentlig av tre deler,

- `width`
- `style`
- `color`

for tykkelse, rammestil og farge, så disse ser dermed slik ut:

```
border-top-width:
border-top-style:
border-top-color:
```

osv.

rem og em

rem og **em** angir størrelse relativt til **font-size** hos hhv. rot (HTML) eller hos seg selv (eller forelderelementet).

```
1em = 1 x gjeldende font-size
2em = 2 x gjeldende font-size
```

Hvis ingen **font-size** er eksplisitt satt, brukes nettleserens standardverdi som basis (ofte er 16px).

%

Prosent er relativ til foreldreelementet.

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

.box {
  width: 50%;
}
```

vw og vh

vw og **vh** står for hhv. vertikal og horisontal viewport.

- 1vw = 1% av skjermes bredde i px
- 1vh = 1% av skjermen bredde i px høyde

```
div {
  width: 50vw;
  height: 100vh;
}
```

Alle typer angivelse kan blandes:

```
div {
  width: 20rem;
  padding: 2em;
  margin: 5%;
  height: 50vh;
}
```

Display

Blokker kan vises under eller ved siden av hverandre. Dette styres med ulike valg for **display**:

- `display: block`
- `display: inline`

- `display: inline-block`

Det fins i tillegg to valg `flex` og `grid` som vi evt. behandler senere.

Betydningen er som følger:

- Blokk-elementer legger seg under hverandre
- Inline-elementer legger seg ved siden av hverandre, men uten egne width/height-settinger
- Inline-block legger seg ved siden av hverandre og kan ha egne width/height-settinger

En `display: block` starter på ny linje og tar hele tilgjengelige bredde (som standard). Den respekterer både `width`, `height`, `margin`, `padding` og `border` som brukeren evt. setter.

```
BLOCK-ELEMENTER

+---------------------------+
|          div              |
+---------------------------+

+---------------------------+
|          div              |
+---------------------------+

+---------------------------+
|          div              |
+---------------------------+
```

`display: inline` gir inline-elementer. De starter ikke nødvendigvis på ny linje, men legger seg ved siden av omgivelsene og flyter inni tekstlinjen (som f.eks. fet skrift gjør i vanlig tekst). Høyde/bredde tilpasses innholdet.

```
+---------+  +---------+  +---------+
|  box 1  |  |  box 2  |  |  box 3  |
+---------+  +---------+  +---------+
```

`display: inline-block` er nokså lik, bare at den respekterer `width`, `height`, `margin` og `padding` fullt ut, dvs. at den kan ha høyde og bredde som skiller fra den umiddelbare omgivelsen.

margin skaper avstand mellom alle disse blokk-typene.

Vi har:

Block-baserte tekstelementer: **div**, **pre** og **p**

Inline-baserte tekstelementer: **code** og **span**

Det er egentlig ikke HTML-standarden som tvinger fram boksegenskapene til disse elementene. Isteden er det nettleseren (kalt *user agent stylesheet* i denne sammenheng) og brukerens CSS-

spesifikasjoner som realiserer dette. I prinsippet kan man endre boksegenskapene til f.eks. **code**, selv om dette ikke er i tråd med godt design.

Normal flyt – blokk:

```
Blokkelementer plasseres vertikalt under hverandre, fyller tilgjengelig bredde og
skyver påfølgende elementer nedover.
```

Normal flyt – inline:

```
Inline-elementer plasseres horisontalt i samme linje som tekst, bruker bare nødvendig
bredde og brytes automatisk til ny linje når det ikke er mer plass.
```

Normal flyt kan endres med `position`, `float`, `flex` og `grid`, som gir alternative måter å plassere elementer på.

Arv

Vi må si litt mer om hva som arves og ikke, samt hvordan arverekkefølgen bestemmes. Arv er altså viktige fordi arvelige egenskaper satt for forfedre kan bli gjeldene for aktuelt element.

Tekst-relaterte egenskaper arves, dvs. egenskaper som

- color
- font-family
- font-size
- font-weight
- line-height
- text-align
- visibility

Boks- og layout-egenskaper arves ikke, dvs. egenskaper som

- margin
- padding
- border
- width
- height
- display
- position
- background
- box-shadow

Man har imidlertid kodeordet/verdien `inherit` som tar verdien fra forelder, uansett om den normalt arves eller ikke. Slik brukes den:

```
div {
  margin: inherit;
}
```

I forlengelsen av alt dette er det også verdt å nevne at det ikke er definert default-verdier for egenskaper i HTML-standarden. Nettlesere (*user agent stylesheet*) setter imidlertid typisk flere av disse (med kun mindre forskjeller mellom ulike lesere).

Når det gjelder arverekkefølge, kan den for tekstelementer typisk se slik ut:

```
html
 └──── body
     └──── main
         └──── div
             └──── p, pre
                 └──── code, span
```

Rekkefølgen er ikke gitt direkte av HTML-standarden, men er en blanding av hvordan rekkefølgen fremkommer av selve HTML-dokumentet man jobber med (hva som ligger inni hva) og HTMLs såkalte *content model*. Kort fortalt har man ulike innholdstyper i HTML:

- **Flow content**: det meste av vanlige elementer
- **Phrasing content**: inline-tekstinnhold
- **Sectioning content**: section, article, nav, aside
- **Heading content**: h1–h6
- **Interactive content**: button, input, href

Og så har man regler som:

- Flow-elementer kan inneholde flow
- Flow-elementer kan inneholde phrasing
- Phrasing kan bare inneholde phrasing

Dette kan være greit å kjenne til, men det er likevel kjappere å forholde seg til følgende tabell, som for hvert tekstelement sier hvilke det kan inneholde og ikke iht. *content model*:

```
Forelder\Barn   div     p     pre   code    span
 div             Y      Y      Y      Y      Y
 p               -      -      -      Y      Y
 pre             -      -      -      Y      Y
 code            -      -      -      Y      Y
 span            -      -      -      Y      Y
```

Vi ser f.eks at **p** kan ligge inni **div**, men ikke omvendt. Videre kan **code** og **span** ligge inni en **pre**, men ikke **div**, **p** eller andre **pre** osv.

Det fins flere tekstlige elementer enn dette, som unummererte og nummererte lister av ulike slag. Vi tar ikke detaljene her, men kort fortalt fins egne beholdere **ol** og **ul** som listeelementeneligger inni:

```
UL (unordered list)   → container for punktliste
 └──── LI               → hvert punkt i listen

OL (ordered list)     → container for nummerert liste
 └──── LI               → hvert punkt i listen
```

Beholderne kan ligge inni **div**, men ikke i **p** eller **pre**. (På den annen side kan **div**, **p**, **pre**, **code**, **span** disse ligge inne **li**, om ønskelig.)

Class

Man kan formatere f.eks. en **div** med:

```
div {
    color: darkblue;
}
```

Dette fungerer, men problemet er da at *alle* **div** endres iht. dette, om man ønsker det eller ikke. Men man kan heldigvis innføre sin egen **div**, f.eks. en kalt **info**, ved CLASS-konstruksjonen i HTML som følger:

```
<div class="info">
    God jul!
</div>
```

Dermed kan man stile alle forekomstene av disse videre uten å påvirke global **div**. Man har flere måter å gjøre dette på, som vi skal se, men vi kan starte med denne:

```
.info {
    color: darkblue;
}
```

Alle forekomster av `<··· class="info">` får blå tekst. Dette er svært nyttig og benyttes over alt. Men CLASS kan utnyttes på flere måter. Man kan nemlig anvende denne flotte, blå info-stilen vår også på andre elementer, f.eks. på **p** ved:

```
<p class="info">
    God påske!
</div>
```

Den ovennevnte CSS-regelblokken vår vil treffe både `<div class="info">` og `<p class="info">`, og teksten blir blåfarget i alle forekomster av begge.

Man kunne vært mer spesifikk og formatert dem separat ved;

```
div.info {
    color: darkblue;
}
```

```
p.info {
    color: blue;
}
```

Dette betyr at man kan gruppere et sett av egenskaper med verdier i en klasse, og anvende den på flere elementer, f.eks. for en mer enhetlig stil eller oppførsel.

La oss se på et eksempel skribenter fort støter på. Ofte konverterer man fra Markdown (med **pandoc**) eller fra Asciidoc (med **asciidoctor**) til HTML. De to første formatene er brukervennlige, men ikke like universelt tilgjengelig for lesing som HTML. La oss si vi har et Asciidoc-dokument som inneholder kodeeksempler, f.eks. en illustrasjon av en `fd`-kommando i bourne-shell. Kodeblokker som `[source,bash]` konverteres av **asciidoctor** til et HTML-elementer av typen

```
<pre>
  <code class="language-bash"> ...
```

Dermed kan man style kodeeksempelet vha.

```
.language-bash {
    color: lightblue;
    background: darkblue;
}
```

eller bedre og mer spesifikt ved

```
code.language-bash {
    color: lightblue;
    background: darkblue;
}
```

Som vi skal se i neste kapittel, har vi også mulighet til å stile det ved

```
pre code.language-bash {
    color: lightblue;
    background: darkblue;
}
```

der vi kun stiler forekomster av `<code class="language-bash">` som ligger inni en **pre** (en såkalt kombinert selektor), samt ved

```
pre .language-bash {
    color: lightblue;
    background: darkblue;
}
```

som stiler alle forekomster av `class="language-bash"` inni en **div** (altså f.eks. også en `<p class="language-bash">` inni en **div**).

Kombinerte selektorer

Kombinerte selektorer er måter å velge HTML-elementer på basert på deres relasjon til andre elementer i dokumentstrukturen. Her ser vi de viktigste kombinasjonene:

```
Selektor    Betydning
A B         B er etterkommer av A
A > B       B er direkte barn av A
A + B       B kommer rett etter A
A ~ B       B er senere søsken av A
```

Det følgende forsøker dermed å stile alle forekomster av **code** under forutsetning at de ligger inni en **pre**:

```
pre code {
  background: lightgray;
}
```

code må ligge inni pre, men ikke nødvendigvis rett etter, så det matcher f.eks:

```
<pre>
  <div>
    Dette er <code>Python</code> for nybegynnere.
  </div>
</pre>
```

Bruker man denne

```
pre > code {
  color: red;
}
```

forlanger man at **code** er direkte barn av **pre**, slik at det bare matcher

```
<pre>
  <code>...</code>
</pre>
```

osv.

pre code har høyere spesifisitet enn bare **code**, fordi den er mer presis. Men den er fortsatt svakere enn en CLASS eller ID.

Moderne nettlesere støtter også **has**-selektor:

```
pre:has(code.language-bash) {
  background: lightgray;
}
```

som formater **pre** kun hvis den inneholder bash-kode (støttes ikke overalt).