Préprocesseurs vs CSS natif : Le match !

KiwiParty
15 juin 2018, Strasbourg

Jonathan Levaillant

Il était une fois CSS (1996)

CSS est un langage descriptif :

1. Impossibilité de créer des **logiques conditionnelles.**
2. Impossibilité de créer des **boucles.**
3. Impossibilité de créer des **fonctions.**

Conséquences :

À terme, le CSS devient vite **complexe**, **répétitif** et **inmaintenable.**

Chapitre 1
L'arrivée des préprocesseurs CSS (2006)

Un préprocesseur CSS, c'est quoi ?

Définition Wikipédia :

Un préprocesseur CSS est un outil (ou programme) informatique permettant de générer dynamiquement des fichiers CSS.

L'objectif est d'améliorer l'écriture de ces fichiers, en apportant plus de flexibilité au développeur web.

Source : https://fr.wikipedia.org/wiki/Préprocesseur_CSS

Préprocesseurs CSS : Popularité

4 principaux acteurs :

Panel de 5 097 développeurs Front-End (mai 2018).

1.	Sass	→	65.33%	+1.84%
2.	PostCSS	→	8.85%	+0.97%
3.	Less	→	6.44%	-3.78%
4.	Stylus	→	2.18%	-0.87%

14.42% des développeurs Front-End n'utilisent pas de préprocesseurs CSS.

Source : https://ashleynolan.co.uk/...css-processors

Préprocesseurs CSS : Fonctionnalités clés

Fonctionnalités programmatiques :

1. Variables : **$variable**
2. Opérateurs mathématiques : **+**, **–**, *****, **/**, **%**
3. Opérateurs relationnels : **<**, **>**, **<=**, **>=**, **==**, **!=**
4. Structures conditionnelles : **@if**, **@else**
5. Boucles : **@each**, **@for**, **@while**
6. Fonctions : **@function**

Préprocesseurs CSS : Fonctionnalités clés

Fonctionnalités structurelles :

1. Imbrication de sélecteurs (nesting).
2. Import de fichiers : **@import**
3. Réutilisation de styles : **@mixin**
4. Héritage : **@extend**, **%placeholder**

Chapitre 2
Préprocesseurs CSS : Les dérives ?

Imbrication de sélecteurs

Définition :

Fonctionnalité permettant d'imbriquer des sélecteurs CSS à l'intérieur d'autres sélecteurs CSS.

Objectif :

Créer des raccourcis d'écriture.

Imbrication de sélecteurs

SCSS

```scss
nav {
  ul {
    li {
      a {
        color: #ff0000;
        &:hover { color: #00ff00; }
        #icon { opacity: .75; }
      }
    }
  }
}
```

CSS

```css
nav ul li a {
  color: #ff0000;
}

nav ul li a:hover {
  color: #00ff00;
}

nav ul li a #icon {
  opacity: .75;
}
```

Imbrication de sélecteurs

SCSS

```scss
nav {
  ul {
    li {
      a {
        color: #ff0000;
        &:hover { color: #00ff00; }
        #icon { opacity: .75; }
      }
    }
  }
}
```

CSS

```css
nav ul li a {
  color: #ff0000;
}

nav ul li a:hover {
  color: #00ff00;
}

nav ul li a #icon {
  opacity: .75;
}
```

Imbrication de sélecteurs

Conséquence N° 1 :
Reproduction du DOM =
Manque de flexibilité.

SCSS

```scss
nav {
  ul {
    li {
      a {
        color: #ff0000;
        &:hover { color: #00ff00; }
        #icon { opacity: .75; }
      }
    }
  }
}
```

CSS

```css
nav ul li a {
  color: #ff0000;
}

nav ul li a:hover {
  color: #00ff00;
}

nav ul li a #icon {
  opacity: .75;
}
```

Imbrication de sélecteurs

Conséquence N° 2 :
Complexité, lisibilité.

SCSS

```scss
        li {
            a {
                color: #ff0000;
                &:hover { color: #00ff00; }
                #icon { opacity: .75; }
            }
            color: #00ff00;
        }
        color: #0000ff;
    }
}
```

CSS

```css
nav ul {
    color: #0000ff;
}

nav ul li {
    color: #00ff00;
}

nav ul li a {
    color: #ff0000;
} ...
```

Imbrication de sélecteurs

Conséquence N° 3 :
Spécificité des sélecteurs.

SCSS

```scss
nav {
  ul {
    li {
      a {
        color: #ff0000;
        &:hover { color: #00ff00; }
        #icon { opacity: .75; }
      }
    }
  }
}
```

CSS

```css
nav ul li a {              /* 0 0 0 4 */
  color: #ff0000;
}

nav ul li a:hover {        /* 0 0 1 4 */
  color: #00ff00;
}

nav ul li a #icon {        /* 0 1 0 4 */
  opacity: .75;
}
```

Quand l'imbrication tourne au cauchemar

Ceci est un vrai projet...

SCSS

```scss
.views-row {
    padding: 0px 0px 8px;
    margin: 0px 0 8px;
    .views-field{
        &.ui-accordion-header{
            display: block;
            width: 100%;
            max-width: 60%;
            margin: 0 auto;
            border: none;
            background: none;
            padding: 6px 0;
            @include font-size(14.3px,17px);
            .ui-icon {
                display: none;
            }
            .field-content {
                a {
                    color: $header-footer-color;
                    letter-spacing: -0.02em;
                    font-weight: bold;
                    max-width: 780px;
                    margin: auto;
                    position: relative;
                    &:before {
                        display: inline-block;
                        width: auto;
                        height: auto;
                        position: relative;
                        content: "\ec1a";
                        @include font-icon-style;
                        font-size: 20px;
                        margin-right: 10px;
                        vertical-align: middle;
                        @include transition(all ease .5s);
                    }
                    &:hover {
                        &:before {
                            color: $hover-icon-color;
                        }
                    }
                }
            }
            @media only screen and (max-width: 768px){
                max-width: 90% !important;
            }
        }
        &.ui-state-active {
            border: none;
            background: none;
            .field-content {
                a {
                    &:before {
```

Quand l'imbrication tourne au cauchemar

Ceci est un vrai projet...

CSS

```
.page-faq .block-faq-block .block-inner .view-content .views-row {
  padding: 0px 0px 8px;
  margin: 0px 0 8px; }
  .page-faq .block-faq-block .block-inner .view-content .views-row .views-field.ui
    display: block;
    width: 100%;
    max-width: 60%;
    margin: 0 auto;
    border: none;
    background: none;
    padding: 6px 0;
    font-size: 17px;
    font-size: 1.18881rem; }
    .page-faq .block-faq-block .block-inner .view-content .views-row .views-field.
      display: none; }
    .page-faq .block-faq-block .block-inner .view-content .views-row .views-field.
      color: #162027;
      letter-spacing: -0.02em;
      font-weight: bold;
      max-width: 780px;
      margin: auto;
      position: relative; }
      .page-faq .block-faq-block .block-inner .view-content .views-row .views-fiel
        display: inline-block;
        width: auto;
        height: auto;
        position: relative;
        content: "\ec1a";
        /* use !important to prevent issues with browser extensions that change fo
        font-family: 'Global-icons' !important;
        speak: none;
        font-style: normal;
        font-weight: normal;
        font-variant: normal;
        text-transform: none;
        line-height: 1;
        /* Better Font Rendering =========== */
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
        font-size: 20px;
        margin-right: 10px;
        vertical-align: middle;
        -webkit-transition: all ease;
        -webkit-transition-delay: 0.5s;
        transition: all ease 0.5s; }
      .page-faq .block-faq-block .block-inner .view-content .views-row .views-fiel
        color: #e01019; }
    @media only screen and (max-width: 768px) {
      .page-faq .block-faq-block .block-inner .view-content .views-row .views-fiel
        max-width: 90% !important; } }
```

Restez simple !

Imbrication de sélecteurs

Conseil :
Limitez-vous aux **pseudo-classes** et/ou **pseudo-éléments.**

SCSS

```scss
.link {
    text-decoration: underline;

    &:focus,
    &:hover,
    &:active {
        color: #ff0000;
    }
}
```

CSS

```css
.link {
    text-decoration: underline;
}

.link:focus,
.link:hover,
.link:active {
    color: #ff0000;
}
```

La directive @extend

Définition :

Fonctionnalité permettant à un sélecteur d'hériter des styles CSS d'un autre sélecteur.

Objectif :

Approche DRY (Don't Repeat Yourself).

La directive @extend

SCSS

```scss
.overlay {
    background-color: #000;
}

.modal {
    background-color: #fff;
}

.modal--dark {
    @extend .overlay;
}
```

CSS

```css
.overlay,
.modal--dark {
    background-color: #000;
}

.modal {
    background-color: #fff;
}
```

La directive @extend

SCSS

```scss
.overlay {
    background-color: #000;
}

.modal {
    background-color: #fff;
}

.modal--dark {
    @extend .overlay;
}
```

CSS

```css
.overlay,
.modal--dark {
    background-color: #000;
}

.modal {
    background-color: #fff;
}
```

La directive @extend

Conséquence N° 1 :
Le **sélecteur étendant** est inséré partout où le **sélecteur étendu** apparaît.

SCSS

```scss
.overlay {
    background-color: #000;
}

.modal {
    background-color: #fff;
}

.modal--dark {
    @extend .overlay;
}
```

CSS

```css
.overlay,
.modal--dark {
    background-color: #000;
}

.modal {
    background-color: #fff;
}
```

La directive @extend

Conséquence N° 2 :
@extend n'accepte pas d'arguments contrairement aux **mixins.**

SCSS

```scss
@mixin size($width, $height) {
    width: $width;
    height: $height;
}

.box {
    @include size(50%, 25%);
    display: flex;
    color: #000;
}
```

CSS

```css
.box {
    width: 50%;
    height: 25%;
    display: flex;
    color: #000;
}
```

La directive @extend

Conséquence N° 3 :
@extend ne peut pas être utilisé dans une media query.

SCSS

```scss
.box {
    display: flex;
    color: #000;
}

@media (min-width: 48em) {
    .card {
        @extend .box;
    }
}
```

CSS

```
.box {
^^^^^
You may not @extend selectors across
media queries.
```

La directive @extend

Conseil :

Utilisez des mixins !

Plus de flexibilité sans les effets négatifs.

SCSS

```scss
@mixin overlay {
    background-color: #000;
}

.modal {
    background-color: #fff;
}

.modal--dark {
    @include overlay;
}
```

CSS

```css
.modal {
    background-color: #fff;
}

.modal--dark {
    background-color: #000;
}
```

Chapitre 3
Le match !

La fonction calc()

Définition :

- La fonction **calc()** permet de réaliser des calculs pour déterminer la valeur d'une propriété CSS.

La fonction calc()

Définition :

- La fonction **calc()** permet de réaliser des calculs pour déterminer la valeur d'une propriété CSS.
- **Il est possible de réaliser des calculs sur des unités différentes.**

Opérateurs mathématiques
vs
Fonction calc()

Opérateurs mathématiques vs Fonction calc()

Opérations simples :

SCSS

```scss
.col {
    width: 100% / 7;
}
```

CSS

```css
.col {
    width: calc(100% / 7);
}
```

Opérateurs mathématiques vs Fonction calc()

Opérations simples :

scss

```scss
.col {
    width: 100% / 7;
}
```

css

```css
.col {
    width: calc(100% / 7);
}
```

Opérateurs mathématiques vs Fonction calc()

Opérations simples :

SCSS

```scss
.col {
    width: 100% / 7;
}
```

CSS

```css
.col {
    width: calc(100% / 7);
}
```

CSS

```css
.col {
    width: 14.2857142857%;
}
```

Opérateurs mathématiques vs Fonction calc()

Opérations complexes (unités différentes) :

SCSS

```scss
.modal {
    position: absolute;
    width: 640px;
    left: 50% - 640px / 2;
}
```

CSS

```css
.modal {
    position: absolute;
    width: 640px;
    left: calc(50% - 640px / 2);
}
```

Opérateurs mathématiques vs Fonction calc()

Opérations complexes (unités différentes) :

SCSS

```scss
.modal {
    position: absolute;
    width: 640px;
    left: 50% - 640px / 2;
}
```

CSS

```css
.modal {
    position: absolute;
    width: 640px;
    left: calc(50% - 640px / 2);
}
```

Opérateurs mathématiques vs Fonction calc()

Opérations complexes (unités différentes) :

SCSS

```scss
.modal {
    position: absolute;
    width: 640px;
    left: 50% - 640px / 2;
}
```

CSS

```css
.modal {
    position: absolute;
    width: 640px;
    left: calc(50% - 640px / 2);
}
```

CSS

```
Incompatible units px and %.
```

Opérateurs mathématiques vs Fonction calc()

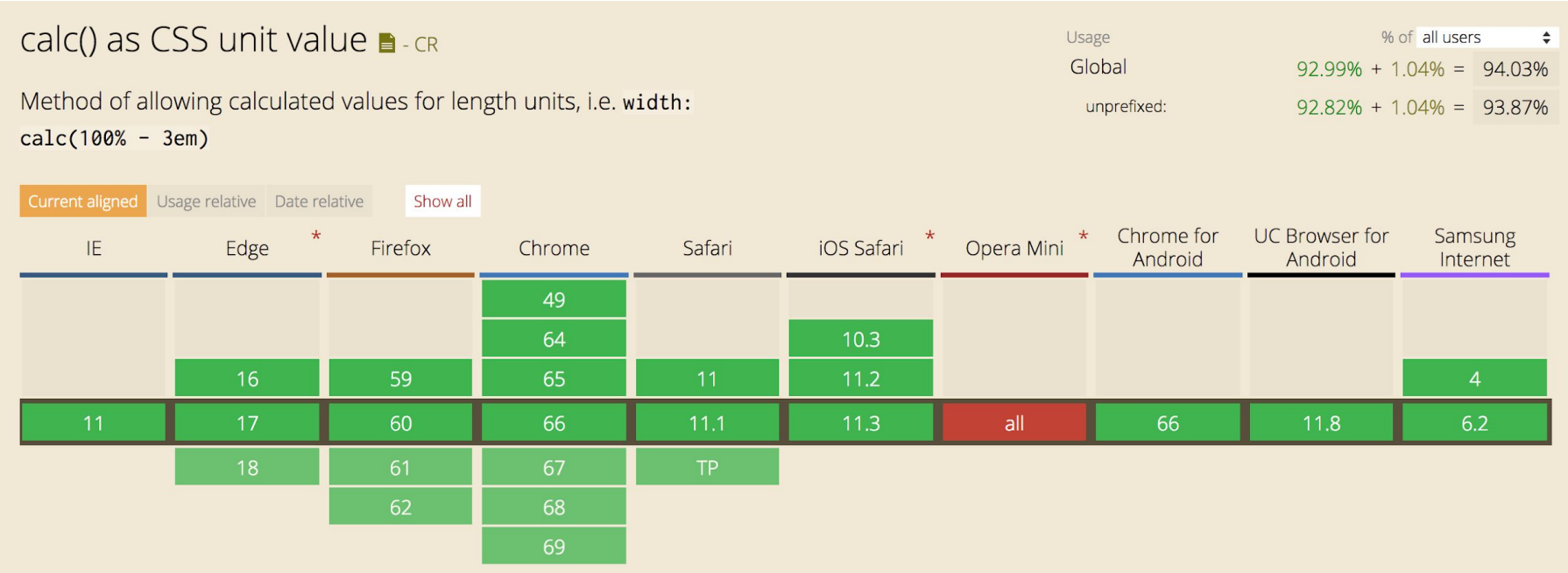
Comparatif :

	Opérateurs mathématiques	Fonction calc()
Unités identiques	✔	✔
Unités différentes		✔
Lisibilité, maintenabilité		✔
Sauvegarde des opérandes		✔

La fonction calc() gagne !

La fonction calc()

Support navigateurs : 94%



Les propriétés personnalisées

Définition :

- Les propriétés CSS préfixées par deux tirets : **--** sont des propriétés personnalisées pouvant contenir une valeur.
- La fonction **var(--)** permet d'accéder aux valeurs de ces propriétés.

Les propriétés personnalisées

Définition :

- Les propriétés CSS préfixées par deux tirets : **--** sont des propriétés personnalisées pouvant contenir une valeur.
- La fonction **var(--)** permet d'accéder aux valeurs de ces propriétés.
- **Les propriétés personnalisées bénéficient de la cascade**.
- **Les valeurs des propriétés personnalisées peuvent être manipulées en JavaScript**.

Variables
vs
Propriétés personnalisées

Variables vs Propriétés personnalisées

SCSS

```scss
$angle: 0deg; $scale: 1;

.clock {
    transform: rotate($angle) scale($scale);
}

.clock--big-ben {
    $scale: 2;
    transform: rotate($angle) scale($scale);
}

.clock--tea-time {
    $angle: 120deg;
    transform: rotate($angle) scale($scale);
}
```

CSS

```css
.clock {
    --angle: 0deg;
    --scale: 1;

    transform: rotate(var(--angle))
               scale(var(--scale));
}

.clock--big-ben {
    --scale: 2;
}

.clock--tea-time {
    --angle: 120deg;
}
```

Variables vs Propriétés personnalisées

SCSS

```scss
$angle: 0deg; $scale: 1;

.clock {
    transform: rotate($angle) scale($scale);
}

.clock--big-ben {
    $scale: 2;
    transform: rotate($angle) scale($scale);
}

.clock--tea-time {
    $angle: 120deg;
    transform: rotate($angle) scale($scale);
}
```

CSS

```css
.clock {
    --angle: 0deg;
    --scale: 1;

    transform: rotate(var(--angle))
               scale(var(--scale));
}

.clock--big-ben {
    --scale: 2;
}

.clock--tea-time {
    --angle: 120deg;
}
```

Variables vs Propriétés personnalisées

CSS

```css
.clock {
    transform: rotate(0deg) scale(1);
}

.clock--big-ben {
    transform: rotate(0deg) scale(2);
}

.clock--tea-time {
    transform: rotate(120deg) scale(1);
}
```

CSS

```css
.clock {
    --angle: 0deg;
    --scale: 1;

    transform: rotate(var(--angle))
               scale(var(--scale));
}

.clock--big-ben {
    --scale: 2;
}

.clock--tea-time {
    --angle: 120deg;
}
```

Variables vs Propriétés personnalisées

CSS

```css
.clock {
    transform: rotate(0deg) scale(1);
}

.clock--big-ben {
    transform: rotate(0deg) scale(2);
}

.clock--tea-time {
    transform: rotate(120deg) scale(1);
}

.clock--big-ben.clock--tea-time {
    transform: rotate(120deg) scale(2);
}
```

CSS

```css
.clock {
    --angle: 0deg;
    --scale: 1;

    transform: rotate(var(--angle))
               scale(var(--scale));
}

.clock--big-ben {
    --scale: 2;
}

.clock--tea-time {
    --angle: 120deg;
}
```

Fonction calc()

+

Propriétés personnalisées

CSS

```css
.grid {
    --gutter: 10px;
    --columns: 12;

    display: flex;
    margin-left: calc(var(--gutter) * -1);
}

.grid__col {
    width: calc(100% / var(--columns) - var(--gutter));
    margin-left: var(--gutter);
}
```

CSS

```css
.grid {
    --gutter: 10px;
    --columns: 12;

    display: flex;
    margin-left: calc(var(--gutter) * -1);
}

.grid__col {
    width: calc(100% / var(--columns) - var(--gutter));
    margin-left: var(--gutter);
}

.grid--lg {
    --gutter: 20px;
}

.grid--2 {
    --columns: 2;
}
```

Variables vs Propriétés personnalisées

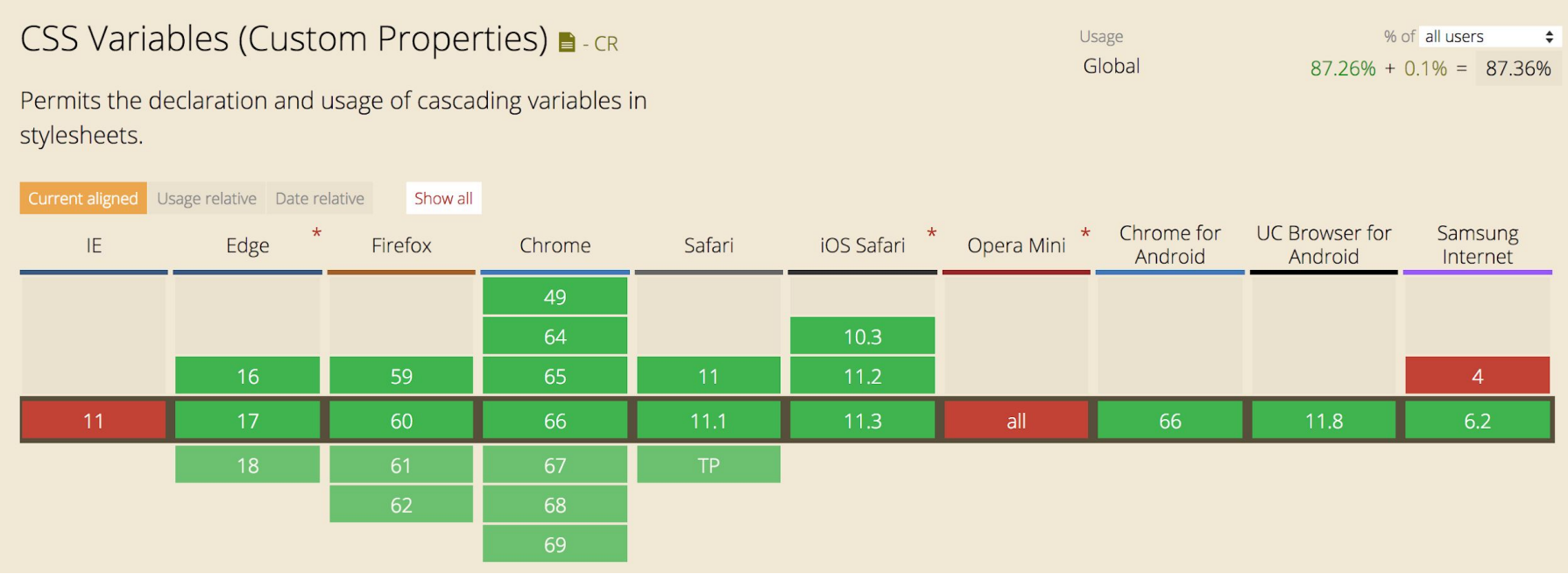
Comparatif :

	Variables	Propriétés personnalisées
Cascade, portée des variables		✔
Manipulation JavaScript		✔
Lisibilité, maintenabilité		✔
Spécificité		✔

Les propriétés personnalisées gagnent !

Variables vs Propriétés personnalisées

Support navigateurs : 87%



La pseudo-classe :matches

Définition :

- La pseudo-classe **:matches()** est une pseudo-classe fonctionnelle permettant de simplifier l'écriture de sélecteurs complexes en les regroupant.

Nesting
vs
Pseudo-classe :matches()

Nesting vs Pseudo-classe :matches()

Combinaison de sélecteurs :

SCSS

```scss
.link {
    &:hover,
    &:active {
        color: #ff0000;
    }
}
```

CSS

```css
.link:matches(:hover, :active) {
    color: #ff0000;
}
```

Nesting vs Pseudo-classe :matches()

Combinaison de sélecteurs :

SCSS

```scss
.link {
    &:hover,
    &:active {
        color: #ff0000;
    }
}
```

CSS

```css
.link:matches(:hover, :active) {
    color: #ff0000;
}
```

Nesting vs Pseudo-classe :matches()

Combinaison de sélecteurs :

SCSS

```scss
.link {
    &:hover,
    &:active {
        color: #ff0000;
    }
}
```

CSS

```css
.link:hover,
.link:active {
    color: #ff0000;
}
```

CSS

```css
.link:matches(:hover, :active) {
    color: #ff0000;
}
```

Nesting vs Pseudo-classe :matches()

Combinaison de sélecteurs (imbrications) :

SCSS

```scss
article, section, aside {
    h1, h2, h3, h4, h5, h6 {
        font-weight: 700;
        color: #ff0000;
    }
}
```

CSS

```css
:matches(article, section, aside)
:matches(h1, h2, h3, h4, h5, h6) {
    font-weight: 700;
    color: #ff0000;
}
```

Nesting vs Pseudo-classe :matches()

Combinaison de sélecteurs (imbrications) :

SCSS

```scss
article, section, aside {
    h1, h2, h3, h4, h5, h6 {
        font-weight: 700;
        color: #ff0000;
    }
}
```

CSS

```css
:matches(article, section, aside)
:matches(h1, h2, h3, h4, h5, h6) {
    font-weight: 700;
    color: #ff0000;
}
```

Nesting vs Pseudo-classe :matches()

Combinaison de sélecteurs (imbrications) :

SCSS

```scss
article, section, aside {
    h1, h2, h3, h4, h5, h6 {
```

CSS

```css
article h1,
article h2,
article h3,
...
aside h4,
aside h5,
aside h6 {
    font-weight: 700;
    color: #ff0000;
}
```

CSS

```css
:matches(article, section, aside)
:matches(h1, h2, h3, h4, h5, h6) {
    font-weight: 700;
    color: #ff0000;
}
```

Nesting vs Pseudo-classe :matches()

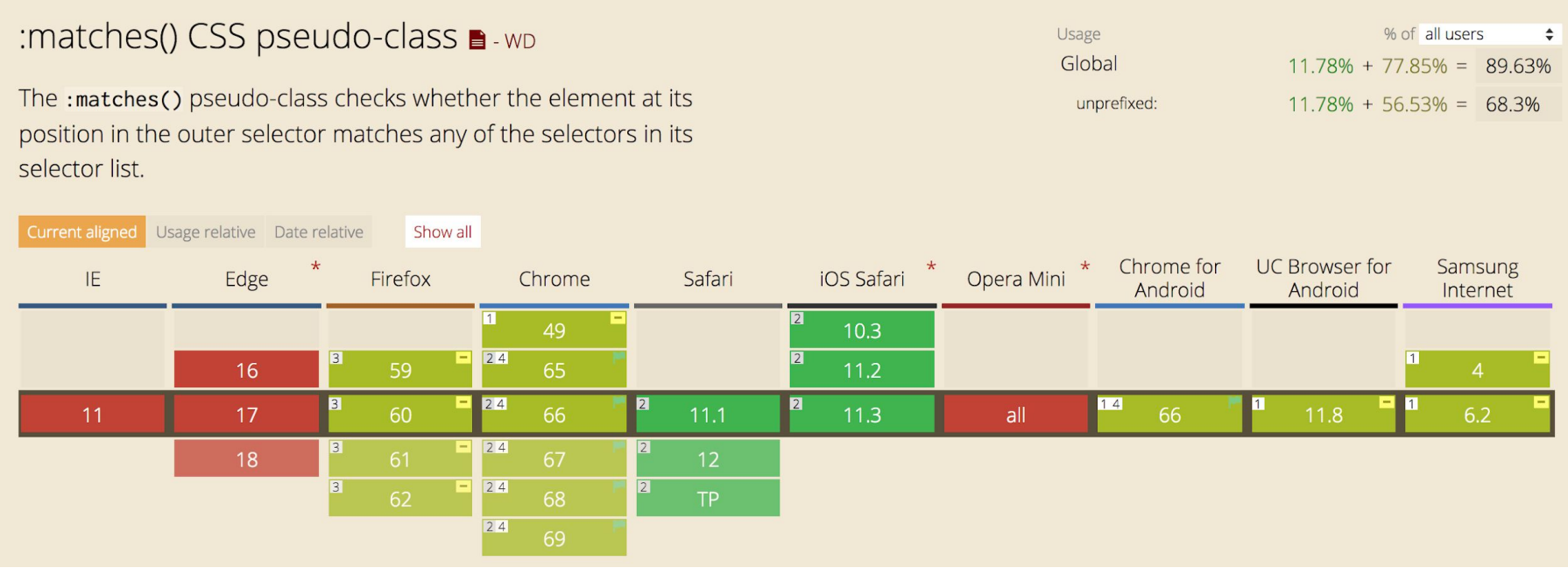
Comparatif :

	Nesting	Pseudo-classe :matches()
Sélecteur de référence (&)	✔	
Combinaison de sélecteurs	✔	✔
Lisibilité, maintenabilité		✔
Poids du CSS généré		✔

La pseudo-classe :matches() gagne !

Nesting vs Pseudo-classe :matches()

Support navigateurs : 90% (Préfixes / Flags)



Chapitre final
Le meilleur des deux mondes ?

Pro Tips

1. Pensez avant tout à la **méthodologie** (BEM, ITCSS, OOCSS, etc.)
2. N'abusez pas du **nesting** : Limitez-vous aux **pseudo-classes**, **pseudo-éléments**.
3. Utilisez la pseudo-classe **:matches** pour simplifier vos sélecteurs.
4. N'utilisez plus la directive **@extend**, préférez les **mixins**.
5. La fonction **calc()** est votre amie !
6. **Variables globales statiques :** Préprocesseurs CSS.
7. **Variables locales dynamiques :** Propriétés personnalisées.
8. **Fonctionnalités programmatiques :** Préprocesseurs CSS.

Merci ! ❤

- Jonathan Levaillant
- Front-End Designer chez <joliCode/>
- Passionné de **CSS** !
- @jlwebart
- **Slides :** http://jonathanlevaillant.fr/2018/kiwiparty.pdf