Technologie Internetu - wykład

Zaawansowany CSS

Flex

- Flex to zestaw reguł CSS pozycjonujących elementy.
- Opieramy się tutaj na rodzicu-kontenerze i elementach-dzieciach.
- Pozycjonowanie jest jednowymiarowe (wiersz lub kolumna).
- Dopasowuje się do zawartości i do okna.
- https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- https://flexboxfroggy.com/

Flex

HTML

```
<div class="wrapper">
    <div class="item">
        pierwszy element
    </div>
    <div class="item">
        drugi element
    </div>
    <div class="item">
        trzeci element
    </div>
</div>
```

CSS

```
.wrapper {
    display: flex;
    flex-direction: row;
    flex-wrap: nowrap;
}
```

Flex

Właściwości elementu rodzica

- **flex-direction** - czy elementy ustawione są w wierszu, czy kolumnie,
- **flex-wrap** - czy mają się przenosić do nowej linii,
- **justify-content** - wyrównanie wzdłuż osi (zgodnej z flex-direction),
- **align-items** - zachowanie elementów prostopadle do osi.

```
.wrapper {
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: stretch;
}
```

Flex

Właściwości elementów dzieci

- **flex-grow** - czy element ma urosnąć by zagospodarować wolną przestrzeń,
- **flex-basis** - wielkość elementu przed rozdzieleniem wolnej przestrzeni,
- **order** - ustawienie innej kolejności niż wynika z ustawienia w kodzie HTML.

```
.item:nth-child(1) {
    flex-grow: 5;
}
.item:nth-child(2) {
    flex-basis: 50%;
}
.item:nth-child(3) {
    order: 1;
}
```

Grid

- Zestaw reguł pozwalający wypozycjonować elementy na stronie.
- Grid = siatka. Mamy wiersze i kolumny (2d).
- Dopasowuje się do zawartości i do okna.
- https://gridbyexample.com/examples/

Grid

HTML

```
<div class="wrapper">
    <div class="item">
        pierwszy element
    </div>
    <div class="item">
        drugi element
    </div>
    <div class="item">
        trzeci element
    </div>
</div>
```

CSS

```
.wrapper{
  display: grid;
  grid-template-columns:
    300px 300px;
  grid-template-rows:
    200px 200px;
  grid-gap: 20px;
}
```

Grid

Właściwości elementu rodzica

- **grid-template-columns** - definiujemy kolumny layoutu i ich wielkość,
- **grid-template-rows** - definiujemy wiersze layoutu i ich wielkość,
- **grid-gap** - przestrzeń między wierszami/kolumnami,
- **grid-template-areas** - można od razu stworzyć całe obszary, do których przypiszemy później pozycję elementów.

```
.wrapper {
    display: grid;
    grid-template-areas:
    “header header header”
    “left main right”
    “footer footer footer”;
}
```

Grid

Właściwości elementów dzieci

- **grid-row, grid-column** - w którym wierszu / kolumnie ma leżeć dany element i na ile wierszy / kolumn się rozciągać (**span**),
- **grid-area** - gdy rodzic posiada nazwane grid-template-areas, możemy podać nazwę obszaru, gdzie dany element ma się ustawić.

```
.item:nth-child(1) {
    grid-row: 1 / span 2;
}
.item:nth-child(2) {
    grid-column: 2 / span 2;
}
nav{
    grid-area: header;
}
```

Transition

- Styl elementu może się zmieniać gdy coś się dzieje, np **:hover**
- Ale taka zmiana nie daje płynnego przejścia.
- Właściwość transition pozwala na animowanie zmiany innej właściwości.

```
div {
    width: 100px;
    height: 100px;
    transition: width 2s;
}

div:hover{
    width: 300px;
}
```

Transform

- Layout strony to prostokąty, ułożone poziomo lub pionowo.
- Teraz możemy wznieść się ponad ten schemat, używając właściwości transform.

```
#item1 {
   transform: rotate(10deg);
}
#item2 {
   transform: skewY(30deg);
}
#item3 {
   transform: scaleY(1.5);
}
```

Zmienne

- Najnowszy CSS pozwala na deklarowanie zmiennych.
- Ułatwia to edycję kodu, gdy dana wartość pojawia się wiele razy.

```
body{
    --main-col: #3399BB;
}

header {
    background: var(--main-col);
}
```

Kalkulacje

- Funkcja calc() wykonuje przeliczenia.
- Można używać wartości w różnych jednostkach, a także zmiennych.

```
body{
    --var1: 100px; --var2: 10%;
}
header {
    height: calc(var(--var1) + var(--var2));
}
```

Gradient

- Tłem strony nie musi być pojedynczy kolor. Może to być gradient.
- Właściwość **linear-gradient** to gradient liniowy;
- Właściwość **radial-gradient** - radialny.
- Można sprecyzować kąt gradientu oraz kolory.

```
body{
    --angle: 120deg; --color1: #3399BB;   --color2: #99BB33;
}
div {
    background-image: linear-gradient(var(--angle),
    var(--color1),
    var(--color2));
}
```

Animacje

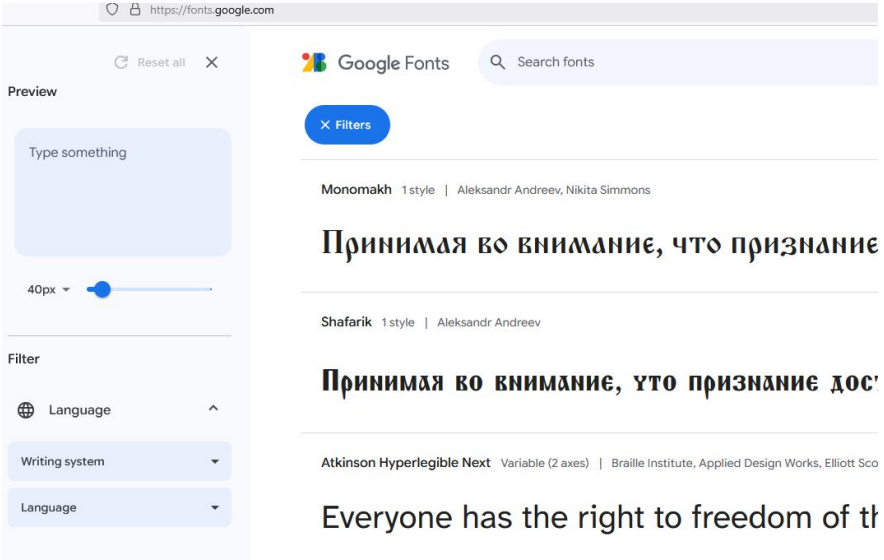
- Proste animacje możliwe są do osiągnięcia przez czysty CSS.
- Reguła **@keyframes** definiuje początkową i końcową wartość animowanej właściwości.
- W stylowanym elemencie wybieramy zdefiniowaną wcześniej regułę oraz ustawiamy szczegóły animacji.

```
@keyframes a1 {
  from {background-color: white}
  to {background-color: black}
}

.item {
  animation-name: a1;
  animation-duration: 5s;
}
```

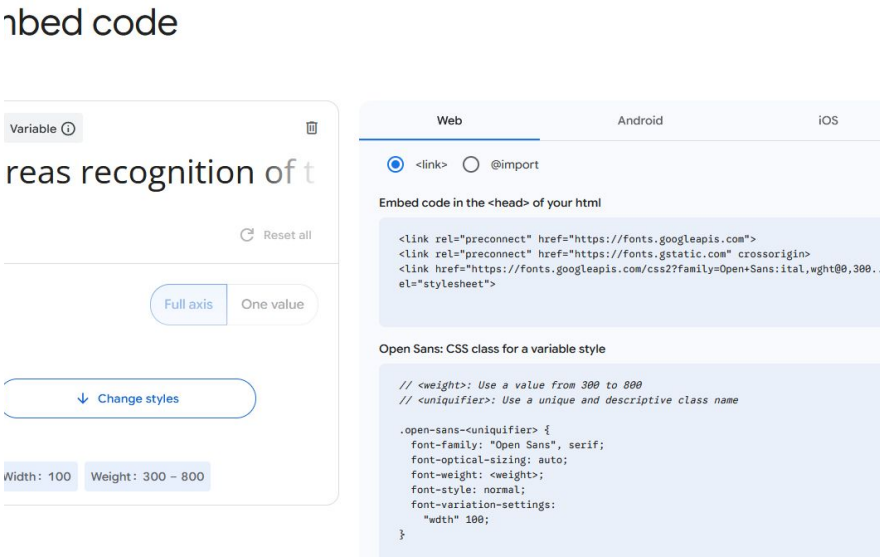
Google Fonts

- Właściwość font-family pozwala określić czcionki użyte w dokumencie.
- Może się jednak okazać, że proponowana przez nas czcionka nie jest zainstalowana na komputerze odbiorcy.
- Rozwiązaniem są zewnętrzne biblioteki do czcionek, takie jak Google Fonts.



Google Fonts

- Znajdujemy ulubioną czcionkę na fonts.google.com
- Wybieramy krój oraz zestaw znaków.
- Opcja **get embed code** wygeneruje nam kod do wklejenia w HTML i CSS.



Responsywność

- Nie tylko komputery wyświetlają strony www.
- Różne urządzenia - różne rozmiary ekranu.
- Wyzwanie - stworzyć stronę, która będzie dobrze wyglądać na każdym.
- RWD - **Responsive Web Design**.

Responsywność

- Każde urządzenie próbuje wyświetlić stronę www najlepiej jak potrafi.
- Czasem może być ona przez to nieczytelna dla użytkownika.
- Pierwsza rzecz w RWD: wiadomość dla przeglądarki, że to my zadbamy o prawidłowy wygląd.

```
<head>
    <title>My website</title>
    <meta name="viewport"
    content="width=device-width, initial-scale=1.0"
    />
</head>
```

Responsywność

Media queries

- Zestawy reguł na różne rozdzielczości.
- Automatycznie wczyta się zestaw pasujący do aktualnych rozmiarów.

```
body { background: red; }

@media screen and
(max-width: 1000px) {
   body { background: blue; }
}

@media screen and
(max-width: 600px) {
   body { background: green; }
}
```

Responsywność

- Strona responsywna powinna zajmować **mało pamięci**.
- Unikamy więc wielkich plików, np. obrazów, filmów.
- Unikamy też animacji (procesor!).
- Unikamy bezwzględnych stylów (np. 100% lepsze od 1000px).
- Jednostki **vh, vw** a także **em** i **rem** sprawdzają się tu.
- Gdzie się da, w ogóle nie stosujmy jednostek.
- **Mobile first**.
- Dużo testów (devtools i prawdziwe urządzenia!).

Organizacja kodu CSS

- Duże aplikacje = dużo kodu CSS.
- Porządek w kodzie = łatwość modyfikacji.
- BEM - przykład metodologii organizacji kodu.
- **Block, Element, Modifier.**

Organizacja kodu CSS

BEM

- **Block** - element blokowy organizujący layout strony.
- **Element** - jakiś element występujący w danym bloku.
- **Modifier** - element przyjmujący konkretny stan (i przez to wygląd).
- Nazewnictwo klas: **.block__element--modifier**

```
.content {
    width: 100%;
}

.content__table {
    background: yellow;
}

.content__table--summary {
    background: blue;
}
```

Minifikacja kodu CSS

- Duże aplikacje = dużo kodu CSS = dużo pamięci.
- Można zmniejszyć wielkość pliku, wyrzucając znaki, które nic nie znaczą.
- Plik na serwerze ma być czytelny dla przeglądarki, a nie człowieka.
- Minifikacja najczęściej stosowana jest do plików CSS i JS.
- Istnieją do tego biblioteki oraz minifikatory online.

Minifikacja kodu CSS

Przed

```
body{
    font-family: serif;
    line-height: 1.5em;
    margin: 0;
}
ul{padding: 0}
#logo img{height: 80px;}
#menu li{
    display:inline-block;
    padding: 0px;
}
```

Po

```
#menu li,ul{padding:0}body{font-family:serif;line-height:1.5em;margin:0}#logo img{height:80px}#menu li{display:inline-block}
```