CSS Animacje Layout

GRKS

Pavlo Zinevych

Agenda

- Display
  - Block
  - Inline
  - Inline-block
- Flexbox
- Gridbox
  - Columns and rows
  - Areas
  - Gaps
- Animacje

Właściwość display (1)

Właściwość display określa zachowanie wyświetlania (typ ramki renderowania) elementu.

W HTML domyślna wartość właściwości wyświetlania jest pobierana ze specyfikacji HTML lub z domyślnego arkusza stylów przeglądarki/użytkownika. Domyślną wartością w XML jest inline, w tym elementy SVG.

```
body {
  display: inline;
}
p {
  display: inherit;
}
```

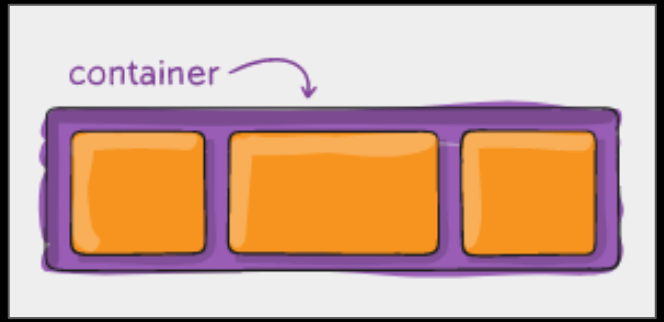
Właściwość display (2)

- **display: inline** - Wyświetla element jako element liniowy (np.). Wszelkie właściwości wysokości i szerokości nie mają wpływu
- **display: block** - Wyświetla element jako element blokowy (jak <p>). Zaczyna się od nowego wiersza i zajmuje całą szerokość
- **display: inline-block -** Wyświetla element jako kontener bloków na poziomie wiersza. Sam element jest sformatowany jako element liniowy, ale można zastosować wartości wysokości i szerokości
- **display: none** - Element jest usunięty
- **display: list-item -** Element zachowuje się jak element
- …

Flexbox - display

Moduł Flexbox Layout (Flexible Box) (rekomendacja W3C z października 2017 r.) ma na celu zapewnienie bardziej wydajnego sposobu układania, wyrównywania i dystrybucji przestrzeni między elementami w kontenerze, nawet jeśli ich rozmiar jest nieznany i/lub dynamiczny (stąd słowo „flex”). **Display** definiuje elastyczny kontener; inline lub block w zależności od podanej wartości. Umożliwia elastyczny kontekst dla wszystkich swoich bezpośrednich dzieci.

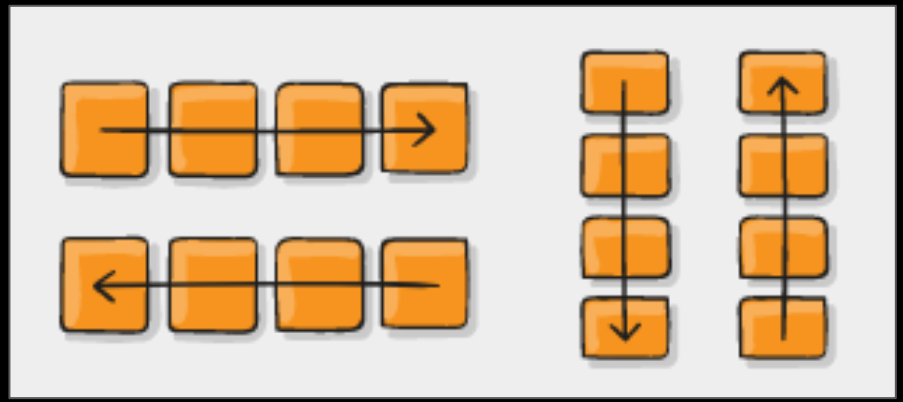
```
.container {
  display: flex; /* or inline-flex */
}
```



Flexbox - flex direction

Flex-direction - stanawia główną oś, definiując w ten sposób kierunek, w którym elastyczne elementy są umieszczane w elastycznym pojemniku. Flexbox to (oprócz opcjonalnego zawijania) koncepcja układu jednokierunkowego. Pomyśl o elementach elastycznych jako układających się głównie w poziome rzędy lub pionowe kolumny.

```
.container {
  flex-direction: row | row-reverse |
column | column-reverse;
}
```



xr_c

Flexbox - flex wrap

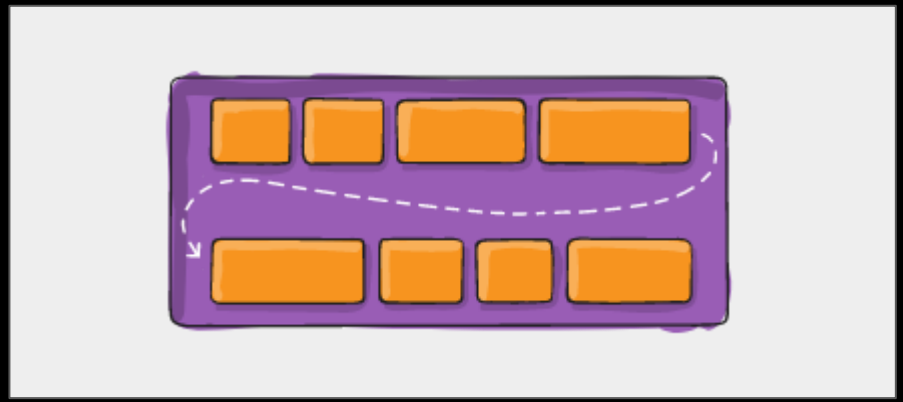
Domyślnie elementy elastyczne będą próbowały zmieścić się w jednym rzędzie. Możesz to zmienić i zezwolić na zawijanie elementów w razie potrzeby za pomocą właściwości **flex-wrap**.

nowrap (domyślnie): wszystkie elastyczne elementy będą w jednym rzędzie

wrap: elastyczne elementy będą zawijane w wiele rzędów, od góry do dołu.

wrap-reverse: elastyczne elementy będą zawijane w wiele rzędów od dołu do góry.

```
.container {
  flex-wrap: nowrap | wrap |
     wrap-reverse;
}
```

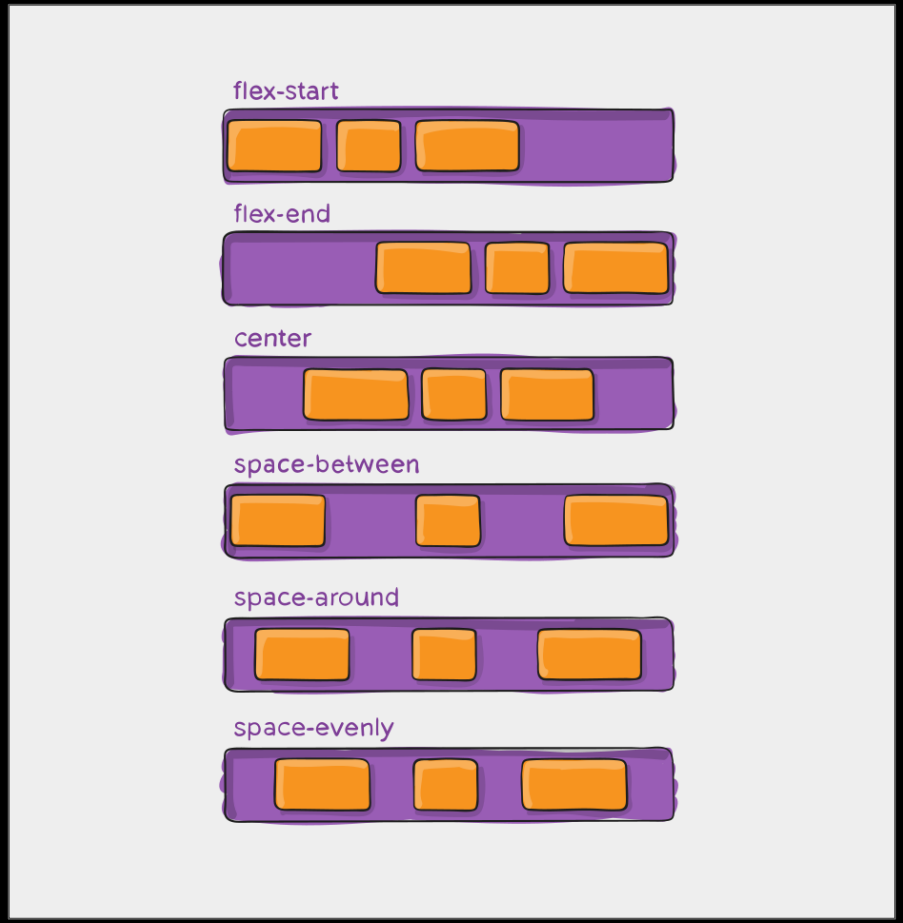


xr_c

Flexbox - justify content (1)

Justify-content definiuje wyrównanie wzdłuż głównej osi. Pomaga rozprowadzać dodatkowe wolne miejsce, gdy wszystkie elementy flex w rzędzie są elastyczne lub nie, ale osiągnęły maksymalny rozmiar. Wywiera również pewną kontrolę nad wyrównaniem elementów, gdy przekraczają one rzęd.

```
.container {
  justify-content: flex-start | ... +
safe | unsafe;
}
```



Flexbox - justify content (2)

- **flex-start** (domyślnie): elementy są pakowane w kierunku początku kierunku flex.
- **flex-end**: przedmioty są pakowane pod koniec kierunku flex.
- **start**: elementy są pakowane w kierunku początku kierunku trybu pisania.
- **end**: elementy są pakowane pod koniec kierunku trybu pisania.
- **left**: przedmioty są pakowane w kierunku lewej krawędzi kontenera, chyba że nie ma to sensu z kierunkiem flex, wtedy zachowuje się jak start.
- **right**: przedmioty są pakowane w kierunku prawej krawędzi pojemnika, chyba że nie ma to sensu z kierunkiem wygięcia, wtedy zachowuje się jak koniec.

Flexbox - justify content (3)

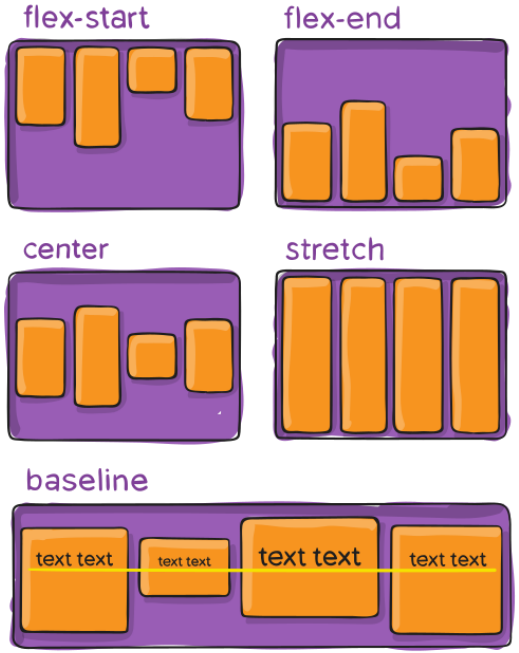
- **center**: elementy są wyśrodkowane wzdłuż linii
- **space-between**: elementy są równomiernie rozmieszczone w wierszu; pierwszy element znajduje się na linii początkowej, ostatni element na linii końcowej
- **space-around**: przedmioty są równomiernie rozmieszczone w linii z równą przestrzenią wokół nich. Zauważ, że wizualnie przestrzenie nie są równe, ponieważ wszystkie przedmioty mają równe miejsce po obu stronach. Pierwszy element będzie miał jedną jednostkę miejsca względem krawędzi kontenera, ale dwie jednostki miejsca między następnym elementem, ponieważ ten następny element ma swoje własne odstępy, które mają zastosowanie.
- **space-evenly**: elementy są rozmieszczone w taki sposób, aby odstępy między dowolnymi dwoma elementami (i przestrzeń do krawędzi) były równe.

Flexbox - justify content (4)

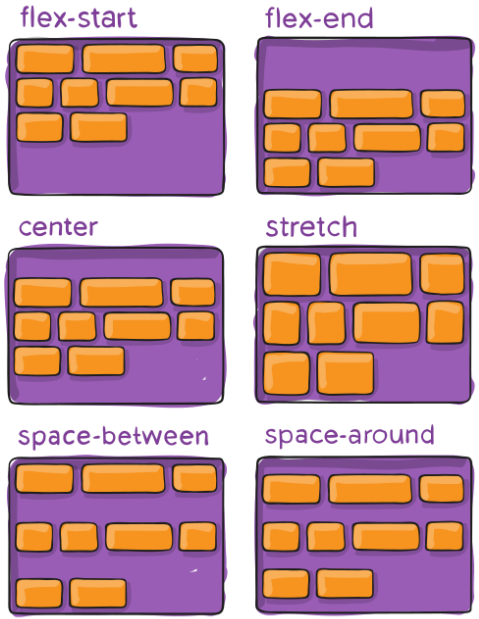
Należy pamiętać, że obsługa tych wartości przez przeglądarkę jest zniuansowana. Na przykład space-between nigdy nie była obsługiwana w niektórych wersjach Edge, a start/end/left/right nie są jeszcze dostępne w Chrome. MDN ma szczegółowe wykresy. Najbezpieczniejsze wartości to flex-start, flex-end i center.

Istnieją również dwa dodatkowe słowa kluczowe, które można sparować z tymi wartościami: safe i unsafe. Korzystanie z safe gwarantuje, że niezależnie od tego typu pozycjonowania, nie można wypchnąć elementu w taki sposób, że renderuje on poza ekranem (np.).

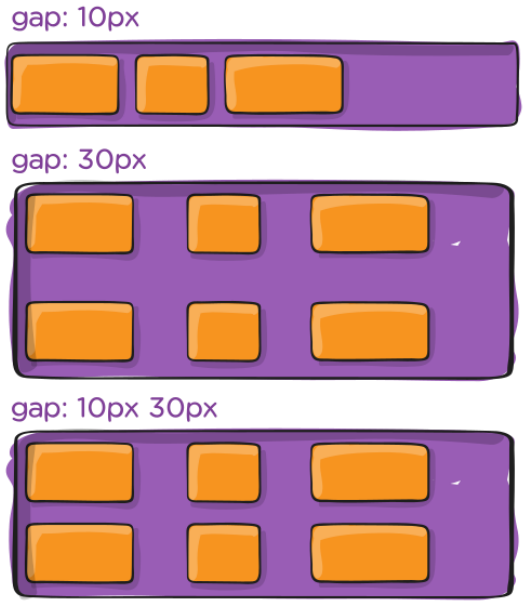
Flexbox - inne właściwości



`align-items`



`align-content`

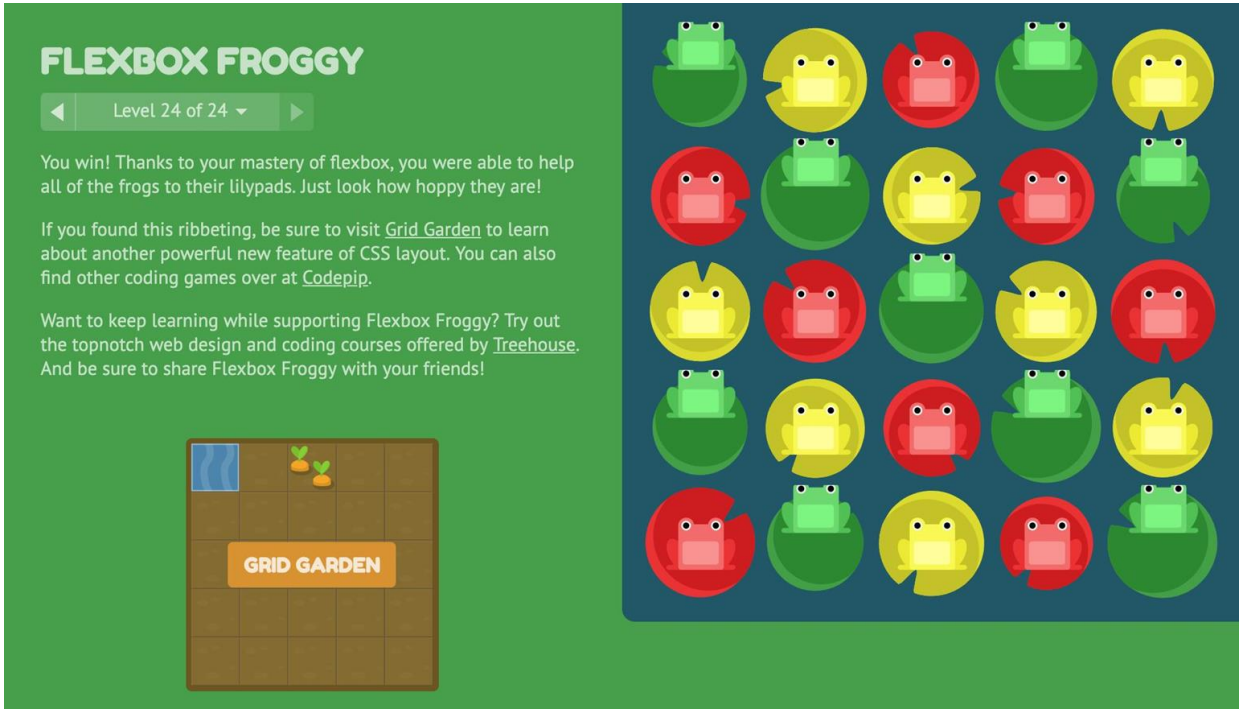


`gap, row-gap, column-gap`

Flexbox - Froggy

Gra do nauki css flexbox.

https://flexboxfroggy.com/



Gridbox

CSS Grid Layout to dwuwymiarowy system układu oparty na siatce, który w porównaniu z jakimkolwiek systemem układu stron internetowych z przeszłości całkowicie zmienia sposób, w jaki projektujemy interfejsy użytkownika (stworzony przed Flexbox).

Display definiuje element jako kontener siatki i ustanawia nowy kontekst formatowania siatki dla jego zawartości.

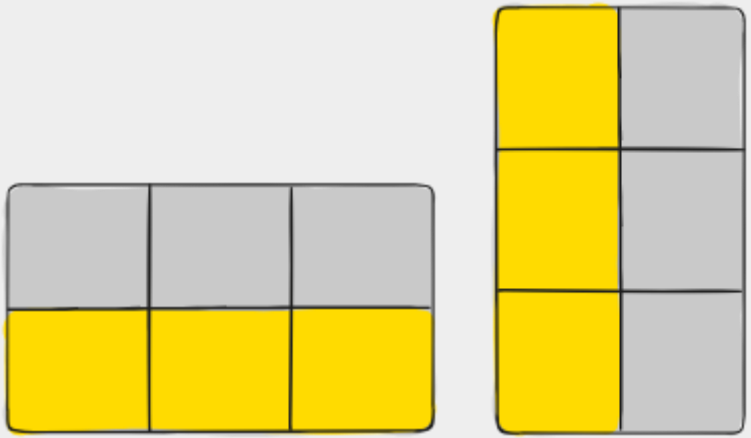
```
.container {

  display: grid | inline-grid;

}
```

Gridbox - grid template columns/rows (1)

Definiuje kolumny i wiersze siatki z rozdzielaną spacjami listą wartości. Wartości reprezentują rozmiar toru, a odstęp między nimi reprezentuje linię siatki.
Wartości:
<track-size> – może być długością, procentem lub ułamkiem wolnego miejsca w siatce przy użyciu jednostki **fr** (więcej na ten temat w DigitalOcean)
<line-name> – dowolnie wybrana nazwa (zawsze w nawiasach kwadratowych), nie są obowiązkowe



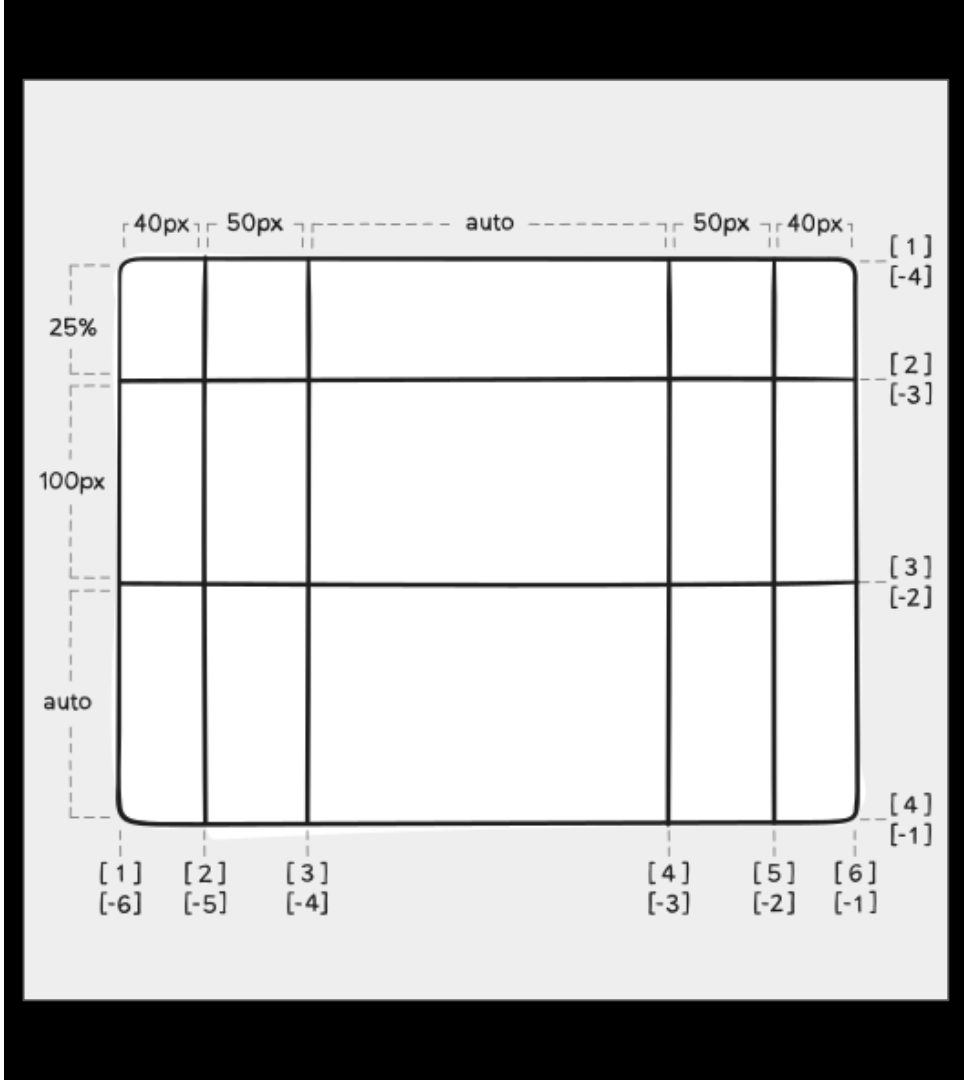
Tor siatki - żółte elementy

Linie siatki - czarne krawędzi elementów

Gridbox - grid template columns/rows (2)

```
.container {
  grid-template-columns: [first] 40px
[line2] 50px [line3] auto [col4-start]
50px [five] 40px [end];

  grid-template-rows: [row1-start] 25%
[row1-end] 100px [third-line] auto
[last-line];
}
```



Gridbox - grid template areas (1)

grid-template-areas - definiuje szablon siatki, odwołując się do nazw obszarów siatki, które są określone za pomocą właściwości grid-area. Powtarzanie nazwy obszaru siatki powoduje, że zawartość obejmuje te komórki. Kropka oznacza pustą komórkę. Sama składnia zapewnia wizualizację struktury siatki.

Wartości:

<grid-area-name> – nazwa obszaru siatki określona w grid-area

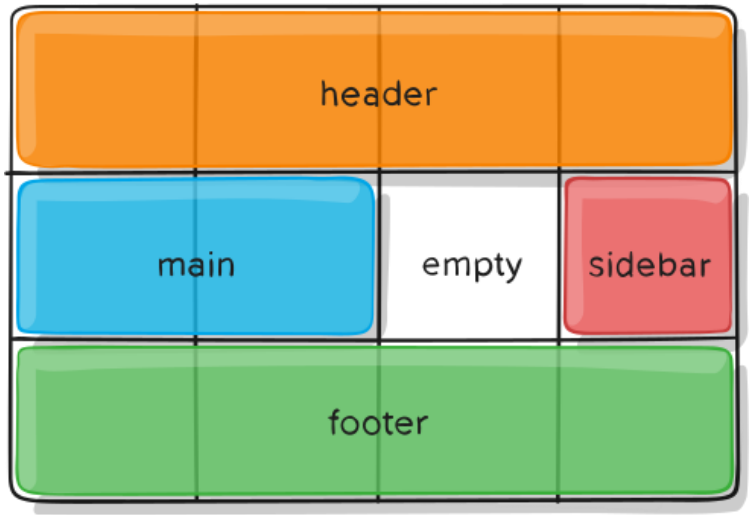
. – kropka oznacza pustą komórkę siatki

none – nie są zdefiniowane żadne obszary siatki

```
.container {
  grid-template-areas:
    "<grid-area-name> | . | none | ..."
                    "...";
}
```

Gridbox - grid template areas (2)

```
.item-a {grid-area: header;}
.item-b {grid-area: main;}
.item-c {grid-area: sidebar;}
.item-d {grid-area: footer;}
```



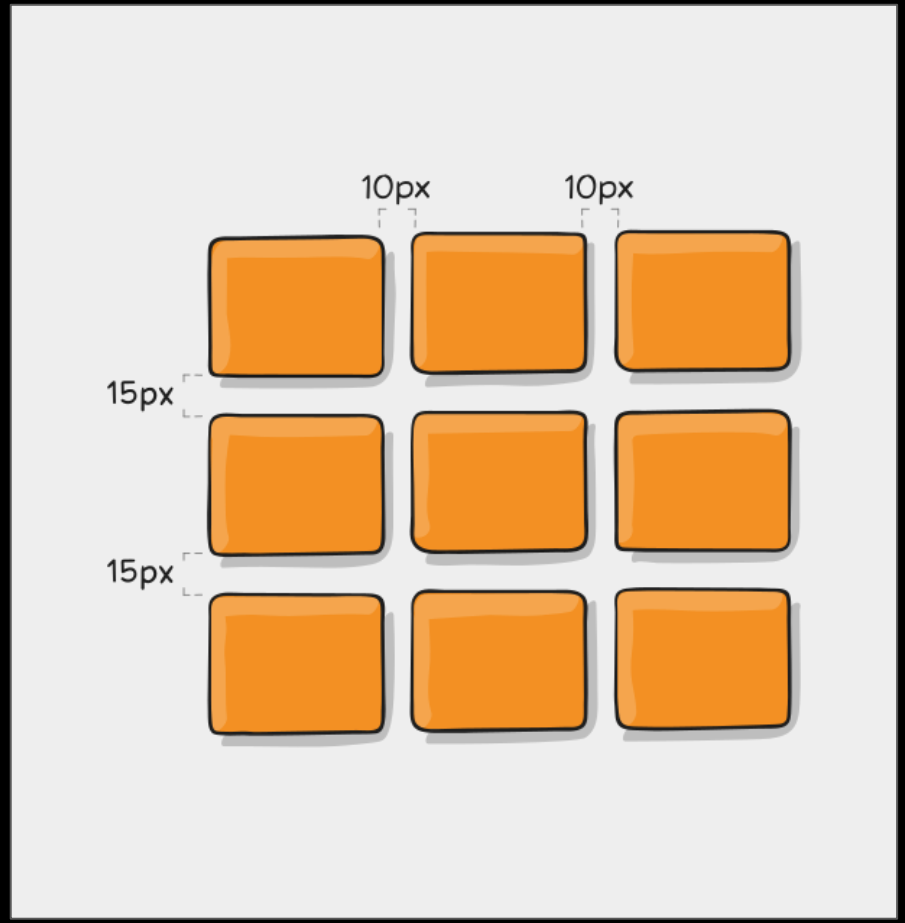

```
.container {
  display: grid;
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
```

Gridbox - Gap (1)

column-gap | row-gap | grid-column-gap | grid-row-gap -

Określają rozmiar linii siatki (ustawienie szerokości odstępu między kolumnami/wierszami).

```
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  column-gap: 10px;
  row-gap: 15px;
}
```



Gridbox - inne właściwości

- `justify-items: start | end | center | stretch`
- `align-items: stretch | start || end | center | …`
- `place-items:`
- `justify-content:`
- `align-content:`
- `grid-auto-columns/rows`
- …

Animacje (1)

Zasada @keyframes

Podanie stylu CSS w obrębie zasady @keyframes sprawi, że animacja będzie stopniowo zmieniać styl z poprzedniego na nowy w pewnym czasie. By animacja działała na elemencie, musi zostać do niego podpięta przy użyciu atrybutu animation-name.

Warto również zaznaczyć, że trzeba podać długość trwania animacji. Defaultowo jest ona ustawiona na 0, co sprawi, że animacja nie odtworzy się w ogóle jeśli ta wartość nie zostanie zdefiniowana.

```
@keyframes nazwa {
            from {
                        background-color:
red;
      }
            to {
                        background-color:
blue;
            }
}
```

```
div {
            width: 200px;
            height: 200px;
            animation-name: nazwa;
            animation-duration:
6s;
}
```

Animacje (2)

Kroki w animacji można również ograniczyć procentowo (gdy animacja jest w 50% wykonana dzieje się x, itd.)

```
@keyframes nazwa {
            0% {
                        background-color: red;
}
            50% {
                        background-color: blue;
            }
            75% {
                        background-color: green;
            }
}
```

W animacjach można oczywiście zmieniać jakiekolwiek właściwości style, również kilka na raz: np. pozycję elementu i kolor.

Animacje - właściwości (1)

- `animation-name` - nazwa identyfikująca animację
- `animation-delay` - określa opóźnienie rozpoczęcia animacji (ile czasu minie zanim animacja zostanie uruchomiona)
- `animation-iteration-count` - określa ile razy animacja ma się wykonać. można podać wartość liczbową, bądź wartość infinite, jeśli chcemy by wykonywała się w nieskończoność
- `animation-direction` - określa czy animacja ma być odtworzona normalnie, na odwrót, czy w cyklach przemiennych. możliwe wartości to:
  - `normal` - domyślna wartość
  - `reverse` - animacja odtworzona “od końca”
  - `alternate` - animacja odtwarza się najpierw normalnie, a potem na od końca do początku, i tak na zmianę
  - `alternate-reverse` - animacja odtwarza się od końca do początku, a potem normalnie, i tak na zmianę

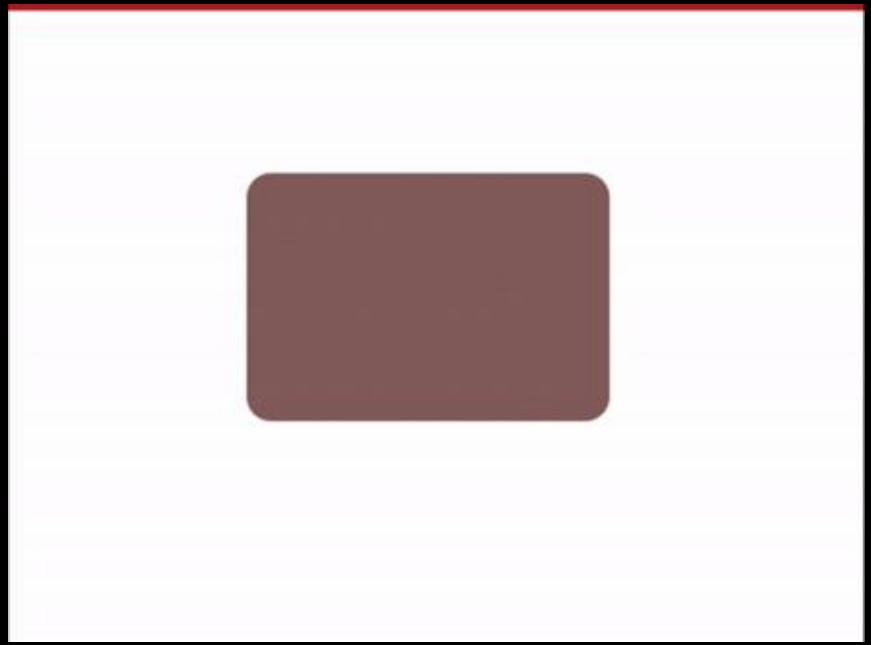
Animacje - właściwości (2)

- `animation-timing-function` - definiuję krzywą prędkości animacji
  - `ease` - wolny start, przyśpieszenie w środku, wolny koniec (wartość domyślna)
  - `linear` - jednakowa prędkość przez całą animację
  - `ease-in` - wolny start
  - `ease-out` - wolny koniec
  - `ease-in-out` - wolny start i koniec
  - `cubic-bezier(n,n,n,n)` - definicja własnych wartości przy pomocy krzywej beziera
- `animation-fill-mode` - animacje css nie wpływają na styl elementu przed wykonaniem się pierwszej klatki oraz po wywołaniu się ostatniej. można to zachowanie nadpisać przy użyciu tego atrybutu. definiuje on styl elementu, gdy animacja nie jest odtwarzana
  - `none` - domyślna wartość, animacja nie zmienia stylu elementu
  - `forwards` - element zachowa style nałożone przez ostatnią klatkę animacji (zależne od animation-direction i animation-iteration-count)
  - `backwards` - element przyjmie style nałożone przez pierwszą klatkę animacji (zależne od animation-direction) i utrzyma je w trakcie działania animation-delay
  - `both` - element będzie działał według zasad zarówno forwards i backwards
- `animation-play-state` - definiuje czy animacja odtwarza się obecnie, czy jest wstrzymana. możliwe wartości to running (domyślne) i paused

Przykład - animacja 1

CSS:

```
div {
  /* animation properties */
  animation-name: my-animation;
  animation-duration: 2s;
  animation-direction: alternate;
  animation-iteration-count: infinite;
  animation-timing-function: linear;
  /* other properties */
  width: 300px;
  height: 100px;
  border-radius: 10px;
  position: absolute;
  left: 0;
  right: 0;
  margin-left: auto;
  margin-right: auto;
}
```

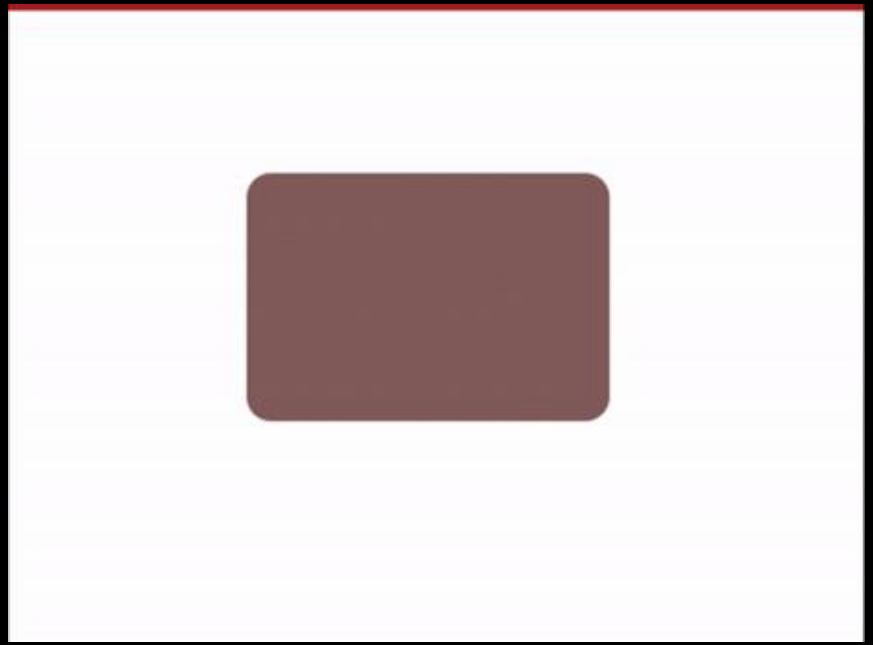


Przykład - animacja 1

```
@keyframes my-animation {
  from {
    background-color: #ff7a59;
    width: 300px;
    top: 10px;
  }
  to {
    background-color: #33475b;
    width: 50px;
    top: 100px;
  }
}
```

HTML:

```
<div></div>
```

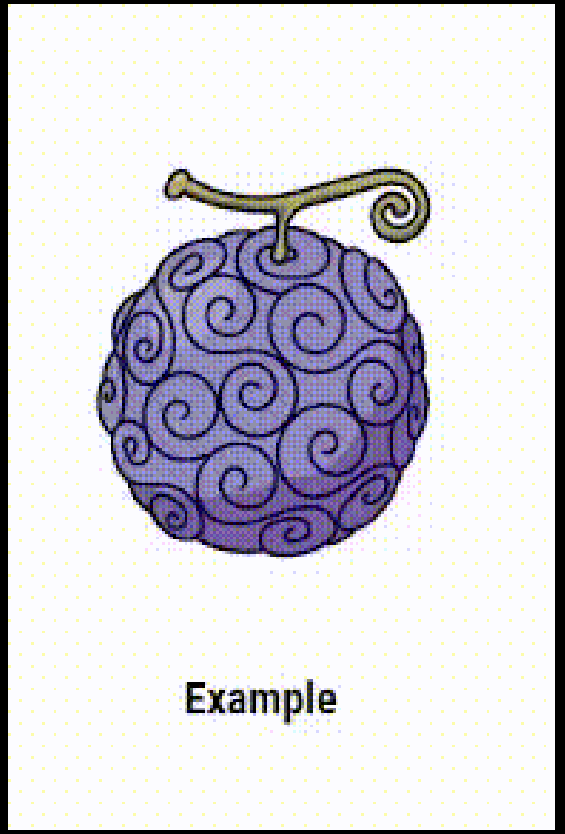


xr_c

Przykład - animacja 2

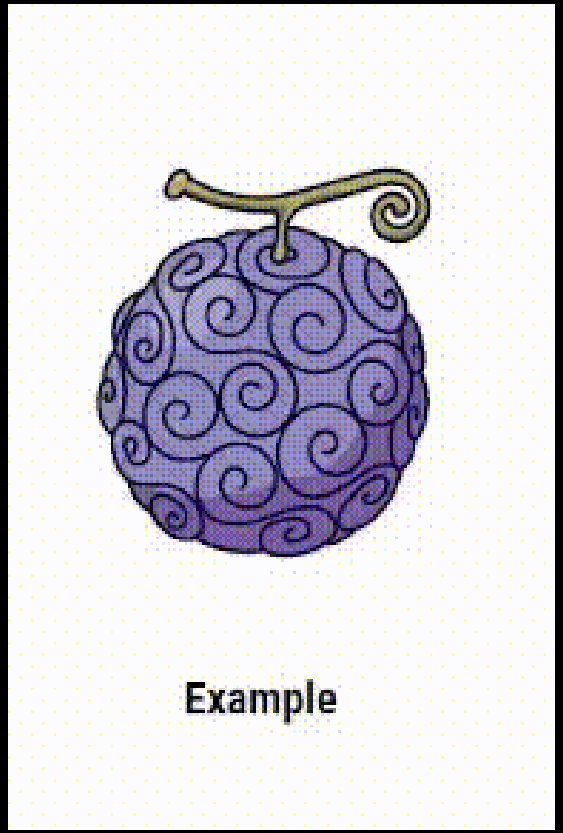
```
<div class="box">
    <img class="gomu" src="gomu.jfif"
            alt="gomu-img">
    <h2>Example</h2>
</div>

.gomu {
    margin-top: 100px;
    animation-name: bounce;
    animation-duration: 2.5s;
    animation-iteration-count: infinite;
    animation-timing-function: linear;
}
```



Przykład - animacje 2

```
 @keyframes bounce {
    0%   { transform: scale(1,1)      translateY(0);
}
    10%  { transform: scale(1.1, 0.7) translateY(0);
}
    30%  {
        transform: scale(0.8,1.1) translateY(-60px);
        rotate: 10deg;
    }
    50%  { transform: scale(1.2,.9) translateY(0); }
    57%  { transform: scale(1,1)    translateY(0);}
    60%  { transform: scale(1,1)    translateY(0); }
    80%  {
        transform: scale(0.8,1.1) translateY(-80px);
        rotate: -10deg;
    }
    97%  {
        transform: scale(1,1) translateY(-10px);
        rotate: 0deg;
    }
    100% { transform: scale(1,1)  translateY(0);}
}
```



Ref

- https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- https://css-tricks.com/snippets/css/complete-guide-grid/
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations
- https://blog.hubspot.com/website/css-animation-examples

Zadanie 1

Stwórz dokumenty HTML i CSS tworzące stronę o dowolnej tematyce według następujących wytycznych bądź rozszerz stronę z poprzednich prac domowych:

- Zapewnij przykładowy układ elementów przy użyciu właściwości display w kontenerze elementów strony. (block/flex/grid)
- W stronie uwzględnij animację przynajmniej dwóch elementów przy użyciu właściwości css animation. Stwórz dwie różne animacje (manipulacja kolorem bądź rozmiarem czy rotacją zależnie od uznania). Skorzystaj z materiałów pomocniczych dostępnych na platformie Teams.
- Pamiętaj o zapewnieniu odpowiedniego podziału dokumentu na elementy semantyczne.
- Strona powinna spełniać standardy dokumentu HTML. Użyj walidatora https://validator.w3.org/ by sprawdzić poprawność swojego dokumentu.