Tworzenie Layout za pomocą Flexbox oraz grid

– porównanie metod

Celem jest stworzenie tego samego efektu za pomocą dwóch różnych technik. Tworzymy ten sam layout strony.

Strona ma wygląd tak jak poniżej:

1. Kod wspólny dla obu przykładów:

Zacznijmy od kodu HTML, który jest wspólny dla obu przykładów. Mamy tutaj div o klasie „container”, który ma czwórkę dzieci, które na potrzeby tego przykładu mają klasy „red”, „blue”, „yellow”, „green”*. W środku każdego dziecka znajduje się paragraf.

```
<div class="container">
  <div class="red">
    <p>czerwony</p>
  </div>
  <div class="blue">
       <p>niebieski</p>
  </div>
     <div class="yellow">
        <p>żółty</p>
     </div>
 <div class="green">
```

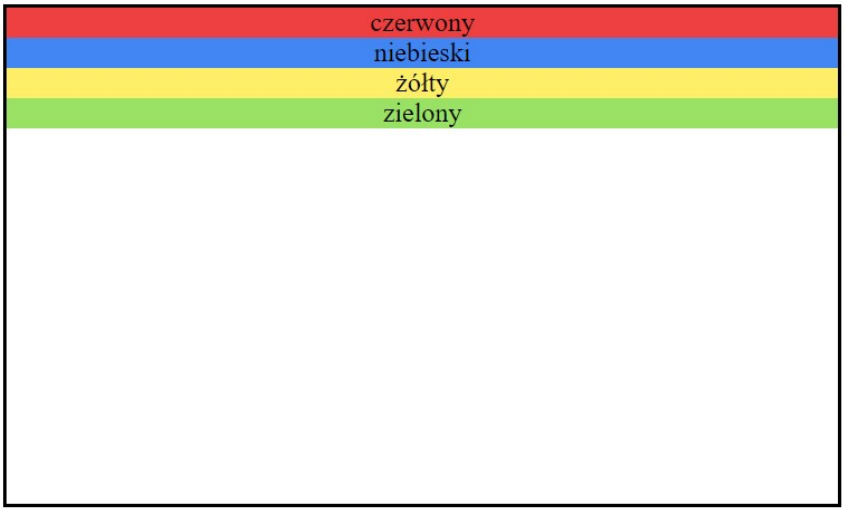
```
    <p>zielony</p>
  </div>
</div>
```

Kod CSS

Na początek dodajemy ogólne style, dla rozróżnienia nadajemy różne kolory poszczególnym dzieciom elementu „container”.

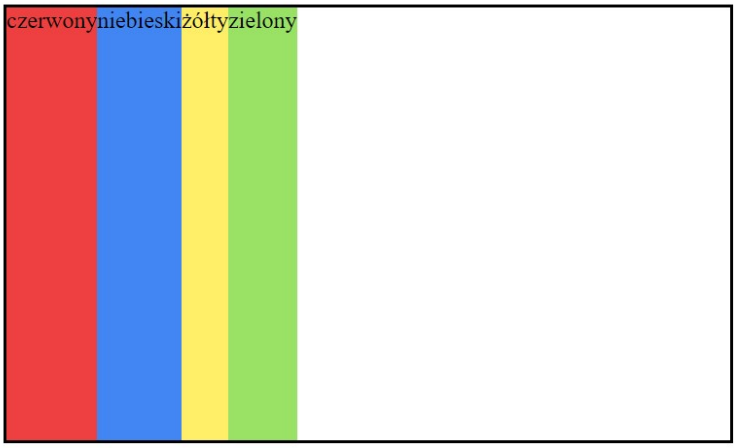
```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

.container {
    width: 500px;
    height: 300px;
    margin: 150px auto 0 auto;
    border: 2px solid black;
}
.red, .blue, .yellow, .green {
        heigth: 100%;
        width: 100%;
        font-size: 25px;
        text-align: center;
}
.red {
    background-color: red;
}
.blue {
    background-color: blue;
}
.yellow {
    background-color: yellow;
}
.green {
    background-color: green;
}
```

Strona wygląd teraz tak:



2. Tworzenie za pomocą FlexBox:

Na sam początek w CSSie nadajemy elementowi o klasie „container” display: flex. W ten sposób cztery divy, które są bezpośrednimi dziećmi elementu „container” zostały uporządkowane jeden obok drugiego, a nie jak było wcześniej jeden pod drugim.

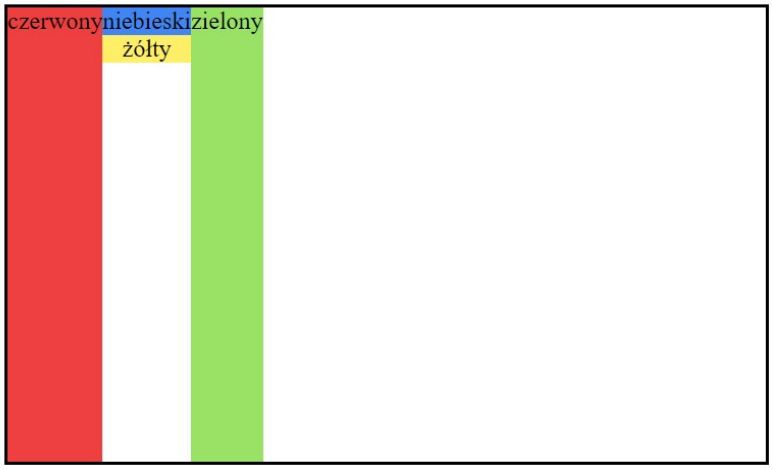


Jest to spowodowane zaaplikowanie domyślnej właściwości jaką jest „flex-direction: row”. Stąd elementu układają się jeden koło drugiego w jednym rzędzie (inne właściwości „flex-direction” to row-reverse, column i column-reverse - można sprawdzić co się stanie gdy wprowadzimy te wartości).

Aby uzyskać zamierzony efekt używając flexboxa musimy nieco zmienić strukturę HTML. Zmierzamy do stworzenia trzech divów, które będą w jednym rzędzie, jak na rysunku wzorcowym (na samej górze). Dodajemy div o klasie „column-wrapper”, w którym znajdą się dwa elementy: „blue” i „yellow”. Teraz div o klasie „container” ma trójkę dzieci: „red”, „column-wrapper” oraz „green”.

```
div class="container">
 <div class="red">
  <p>czerwony</p>
 </div>
 <div class="column-wrapper">
  <div class="blue">
   <p>niebieski</p>
  </div>
  <div class="yellow">
   <p>żółty</p>
  </div>
</div>
 <div class="green">
  <p>zielony</p>
 </div>
</div>
```

Teraz nasz układ wygląda tak:



Teraz musimy jeszcze dodać style na elemencie „column-wrapper”. Skorzystamy z właściwości jaką jest „flex”. Dzięki temu będziemy mogli rozdysponować przestrzeń rodzica za pomocą procentowych wartości. Chcemy, żeby div o klasie „column-wrapper” zajmował połowę szerokości swojego rodzica, a pozostałe elementy zajmowały po 25% szerokości. Modyfikujemy wiec css dopisując:

```
.column-wrapper {
   flex: 0 1 50%
}
.red {
```

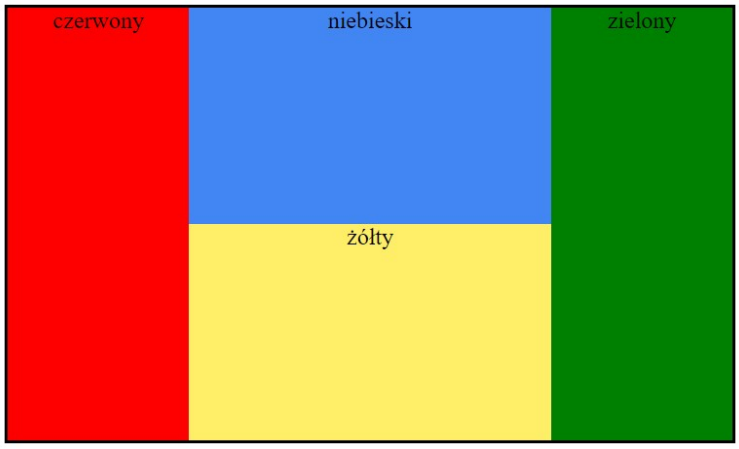
```
  background-color: red;
  flex: 0 1 25%;
}
.green {
  background-color: green;
  flex: 0 1 25%;
}
```

Teraz nasza strona wygląda tak:

Stylujemy dzieci elementu „column-wrapper”. Chemy żeby każde dziecko zajmowało połowę wysokości rodzica.

```
.blue, .yellow {
  height: 50%;
}
```

I mam y efekt końcowy



3. Tworzenie za pomocą CSS Grid :

Ten sam efekt, ale z użyciem CSS Grid. Wracamy do naszego podstawowego kodu. Mamy rodzica o klasie „container” oraz czwórkę dzieci: div-y o klasach „red”, „blue”, „yellow” i „green”. W przypadku CSS Grid nie będziemy tworzyć dodatkowych elementów w HTML, jak było w przypadku Flexboxa (div o klasie „column-wrapper”). CSS Grid pozwala na opisanie struktury dokumentu, bez potrzeby dodatkowych div-ów, które mają na celu jedynie wspomóc jego stylowanie.

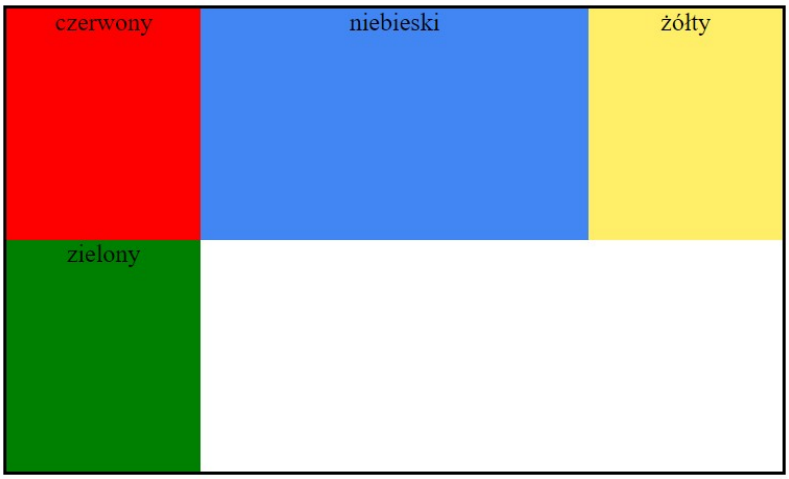
Podstawowe style już mamy, czas przejść do definiowania CSS Grid. Nadajemy elementowi o klasie „container” display: grid. W efekcie elementy zostały ułożone jeden nad drugim oraz wypełniły całą wysokość rodzica.

Jak pamiętamy z zajęć CSS Grid można porównać do macierzy. Możemy definiować kolumny oraz wiersze. Tutaj na poniższej grafice demontuję jak musimy podzielić nasz „container”, żeby uzyskać zamierzony efekt.
Należy zdefiniować trzy kolumny oraz dwa wiersze. Element „red” będzie zajmować dwie komórki (całą pierwszą kolumnę), element „green” również będzie zajmować dwie komórki (całą ostatnią kolumnę). Niebieski to górna środkowy wiersz. Żółty środkowy dolny. Skoro wiemy ile wierszy i kolumn jest potrzebne czas na zdefiniowanie ich wielkości. CSS Grid wprowadza nową jednostkę jaką jest fr czyli „fraction of available space”.

Do diva „container” dodajemy następujące style:

```
.container {
   display: grid;
   grid-template-columns: 1fr 2fr 1fr;
   // to będą trzy kolumny, ich szerokość będzie do siebie w stosunku 1:2:1
   grid-template-rows: 1fr 1fr;
   // to będą dwa wiersze, ich wysokość będzie do siebie w stosunku 1:1
}
```

Po dodaniu tego kodu każdy element zajął tylko jedną komórkę w Grid. Zostały dwie puste komórki.



Aby to zmienić możemy na przykład zdefiniować grid-areas oraz grid-template-area. Grid-area nadawany jest na dzieciach elementu, który ma display grid.

Definiowanie grid-template-area. Musimy zdefiniować, które elementy będą zajmować które i ile komórek. Na elemencie „container” dodajemy:

```
grid-template-areas:
"red blue green"
"red yellow green";
```

Ostatecznie do elementu „container” zostały dodane takie style:

```
.container {
   display: grid;
   grid-template-columns: 1fr 2fr 1fr;
   grid-template-rows: 1fr 1fr;
   grid-template-areas:
   "red blue green"
   "red yellow green";
}
```

Teraz definiujemy grid-area w poszczególnych divach.

```
.red {
   background-color: red;
  grid-area: yellow;
```

```
}
.green {
   background-color: green;
 grid-area: green;
}
.blue {
  background-color: #4286f4;
  grid-area: blue;
}
.yellow {
  background-color: #ffef68;
  grid-area: yellow;
}
```

Efekt końcowy: