**Czym jest flex, grid?**

Przez wiele lat programiści stron internetowych mieli bardzo dużą trudność w pozycjonowaniu elementów. Wiele takich sytuacji wymagało stosowania tzw. "hacków" czyli obchodzenia typowego zastosowania pewnych rozwiązań w celu uzyskania zamierzonego efektu. To były tricki, a nie w pełni stabilne, natywne rozwiązania.  
Dużo zmieniło się od czasu wejścia CSS w wersji trzeciej. To on przyniósł długo wyczekiwane rozwiązania mające ułatwić naszą pracę.  
Flex oraz grid to sposoby na układanie elementów np zgodnie z designem, który został do nas dostarczony.

**Co lepsze? Grid czy Flex?**

Nie można odpowiedzieć na to pytanie, bo oba rozwiązania mają swoje wady i zalety, są stosowane w zaleźności od potrzeb.

Zaletą Grida jest stworzenie sztywnych ram naszego layoutu (czyli wyglądu strony, planu) i automatyczne dopasowywanie ich, w zależności od ilości treści.  
Flex jest bardzo elastyczną metodą i pozwala nam na nieco więcej "kombinowania", ale używany nierozważnie może doprowadzić do błędnego wyświetlania zawartości strony.

**Flex - wprowadzenie**

Pamiętacie parametr CSS display? To w nim będziemy decydować czy element ma być elementem flexowy bądź gridowym.

.box-container {

display: flex;

}

Co nam to na początku daje? To, że divy wewnątrz będą układały się obok siebie. Są dziećmi naszego box-container więc z założenia chcemy je jakoś uporządkować.  
Powiedzmy, że w *box-container* chcemy żeby znalazły się 3 boxy.

* Sam container ma mieć 200px wysokości i 100% szerokości. Ma żółte tło.
* Boxy to divy o rozmiarze 100x100 pixeli, mają niebieskie tło.
* Chcemy żeby te 3 boxy ułożyły się w równych odległościach od siebie, wykorzystując całą dostępną szerokość box-containera
* Chcemy żeby w pionie zostały wyśrodkowane względem box-containera

Warto wyróżnić kilka najczęściej używanych parametrów flexowych:

* justify-content - decyduje o tym jak nasze elementy będą się układać w poziomie
* align-items - decyduje o tym jak nasze elementy będą się układać w pionie
* flex-wrap - decyduje o tym czy nasze elementy mają się załamywać do nowej linii kiedy będzie ich "za dużo"

Nasze pliki wyglądałyby wtedy w następujący sposób:

<div class="box-container">

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

</div>

.box-container {

display: flex;

justify-content: space-between;

align-items: center;

height: 200px;

background: yellow;

}

.box {

width: 100px;

height: 100px;

background: blue;

}

Jeżeli chcielibyśmy aby elementy miały dookoła siebie równe przerwy (żeby skrajne z lewa i prawa nie były przyklejone zaraz do krawędzi kontenera)  
Wystarczyłoby zmienić jeden parametr:

justify-content: space-evenly;

Flex daje nam też możliwość ustalenia kolejności układania elementów. Na przykład zamiast domyślnie układać się w wierszu (horyzontalnie), możemy kazać elementom układać się jeden pod drugim:

.box-container {

flex-direction: column

}

Jeżeli chcielibyśmy zastosować jakieś indywidualne zasady pozycjonowania dla pojedynczego elementu, to skorzystamy z kolei z align-self.  
Nasze boxy były wyśrodkowane ale gdybyśmy chcieli żeby jeden był przyklejony do dołu kontenera, to możemy przypisać mu klasę modyfikatora, w której zastosujemy wspomniany parametr:

.box\_\_bottom {

align-self: flex-end;

}

Jest bardzo dużo zastosowań flexa i tak naprawdę ogranicza nas tylko nasza wyobraźnia. Polecamy zapoznać się z kompletnym tutorialem popularnej strony [CSS-Tricks](https://css-tricks.com/snippets/css/a-guide-to-flexbox/), w którym możecie zobaczyć wszystkie dostępne parametry flexowe wraz z wizualizacją ich działania!  
[CSS-Tricks](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)

**Zadanie domowe - zrób, nim przejdziesz dalej!**

Wyobraźmy sobie najpopularniejszy typ nawigacji na stronie. Co widzimy? Pewnie w większości logo po lewej stronie i linki do odpowiednich sekcji po prawej stronie. Może jest to już nieco zbyt oczywiste, ale równiez bardzo intuicyjne dla wielu użytkowników. Twoim zadaniem będzie więc zrobienie:

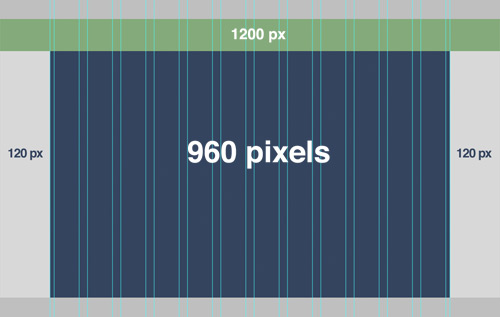
* Nawigacji, o szarym tle, która zajmuje całą szerokość przeglądarki, ma padding z każdej strony na 2%, a w środku znajdą się 2 elementy: logo jako <img> (wybierz dowolne) o szerokości 50px; oraz pozioma lista z linkami (max 4).

Logo powinno być przyklejone do lewej strony, a linki powinny wyświetlać się po prawej. Wysokość nawigacji dostosuje się automatycznie do treści.

* Upewnij się, że logo i nasze linki znajdują się na środku pionie.
* Nie używaj float, tylko stosuj techniki *flex*.

**Rozszerzenie: Flex i Grid w CSS**

Grid**- wprowadzenie**

Czym w ogóle jest grid? Czy to tylko zagadnienie CSS'owe, czy stoi za tym coś więcej? Grid istniał w webdesignie na wiele lat przed wprowadzeniem tej funkcji w CSS.  
Samo pojęcie grida sprowadza się do podzielenia projektu strony/aplikacji na kolumny i wiersze. Zwykle jest to 12 kolumn, rzadziej 8, 6.  
  
  
  
  
Po co się to robi? Ma to związek z zagadnieniem RWD (*Responsive Web Design*), o którym powiemy na następnych lekcjach. W dużym skrócie - łatwiej stworzyć responsywną, elastyczną stronę trzymając się proporcji zamiast stałych wartości.  
Na przykład jakiś element ma zająć 4 z 12 dostępnych kolumn na szerokość - czyli jedną trzecią. Podając to w pixelach bylibyśmy ograniczeni do konkretnych wyświetlaczy, a programista mógłby się nie domyślić tego co w planach miał designer.

**Podstawowe parametry Gridowe**

Analogicznie do zastosowania flex'a, oczywiście w tym przypadku też musimy zacząć od ustawienia odpowiedniej wartości parametrowi display.

.box-container {

display: grid;

}

W naszym przykładzie spróbujemy zrobić prostą siatkę 9x9.  
Będziemy chcieli wyświtlić 9 identycznych, niebieskich, kwadratowych boxów - po 3 w jednym rzędzie. Taki schemat moglibysmy wykorzystać na przykład w grze kółko-krzyżyk albo memory.  
  
Aby dało się je od siebie rozróżnić, ustawimy między nimi przestrzeń - tzw. gap o rozmiarze 1%.

<div class="box-container">

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

</div>

.box-container {

display: grid;

width: 500px;

height: 500px;

grid-template-columns: 33% 33% 33%;

grid-template-rows: 33% 33% 33%;

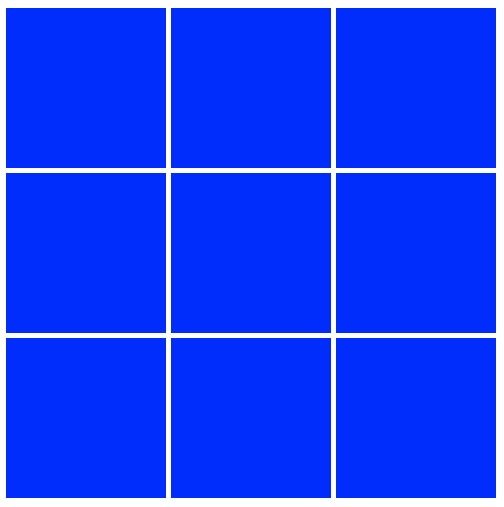
grid-gap: 1%;

}

.box {

background: blue;

}

Efekt?  
  
  
  
Przeanalizujmy podany kod.

grid-template-columns: 33% 33% 33%;

grid-template-rows: 33% 33% 33%;

grid-gap: 1%;

Te parametry sprawiają, że:

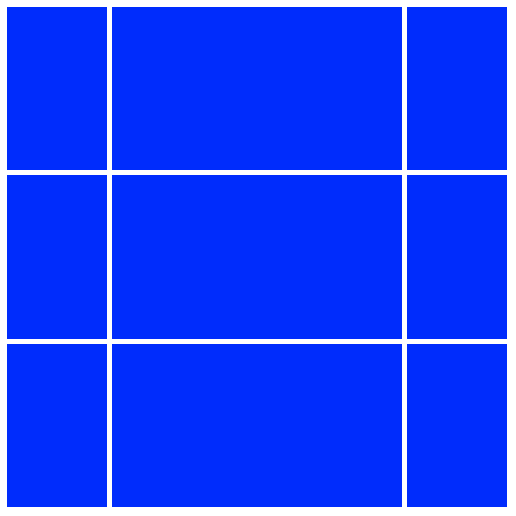
* ustalamy szerokości kolumn - 3 kolumny po 33%, bo chcemy żeby każdy zajął 1/3 szerokości całego kontenera, który w naszym przykładzie ma 500x500 pixeli;
* ustalamy wysokość wierszy - 3 wiersze po 33%, bo chcemy żeby każdy zajął 1/3 wysokości całego kontenera
* ustalamy przerwę między tymi elementami - aby poszczególne "kafelki" były od siebie rozróżnialne, a nie zlały się w całość

Inaczej niż zazwyczaj, tutaj ta przestrzeń pomiędzy elementami dzieje się automatycznie. Chociaż mamy wykorzystane 99% na same kafelki (3 x 33%) to zmieściliśmy 2%.  
  
Może chcielibyśmy stworzyć sekcje z bocznymi menu dla każdej z nich?

grid-template-columns: 20% auto 20%;

grid-template-rows: auto;

grid-gap: 1%;

W tym przypadku mówimy CSS'owi, że pierwsza i ostatnia kolumna mają mieć po 20%, a środkowa ma wykorzystać automatycznie pozostałe miejsce. Tym razem uznaliśmy też, że nasza sekcja może mieć sporo treści i ustalanie jej wysokości na sztywno to zły pomysł, dlatego w grid-template-rows podaliśmy wartość auto. To znaczy, że wiersze dostosują swoją wysokość automatycznie, uwzględniając ich zawartość.  
  
Efekt?  
  
  
  
Przejdźmy jednak do największego "bajeru". Czyli definiowania tego grida w sposób bardzo intuicyjny i obrazowy - ale też przydatny do bardziej życiowych sytuacji.  
Powiedzmy, że nasza strona ma mieć następującą "siatkę" - czyli grid layout. Zdecydowaliśmy, że kolumn będzie 6:

* Nawigacja ma mieć 100% szerokości
* Poniżej ma się znaleźć boczne menu na 2/6 szerokości
* Reszta tego wiersza to artykuły
* Poniżej chcemy wyświtlić 2 reklamy naszych produktów, oba po 50% szerokości
* Na samym dole mamy stopkę na 100% szerokości strony

Oto jak to zapiszemy:

<div class="container">

<div class="nav">nav</div>

<div class="side-nav">side-nav</div>

<div class="content">content</div>

<div class="ad1">ad1</div>

<div class="ad2">ad2</div>

<div class="footer">footer</div>

</div>

.nav {

grid-area: nav;

}

.side-nav {

grid-area: side-nav;

}

.content {

grid-area: content;

}

.ad1 {

grid-area: ad1;

}

.ad2 {

grid-area: ad2;

}

.footer {

grid-area: footer;

}

.container {

display: grid;

grid-template-columns: auto;

grid-template-rows: auto;

grid-template-areas:

"nav nav nav nav nav nav"

"side-nav side-nav content content content content"

"ad1 ad1 ad1 ad2 ad2 ad2"

"footer footer footer footer footer footer";

text-align: center;

grid-gap: 10px;

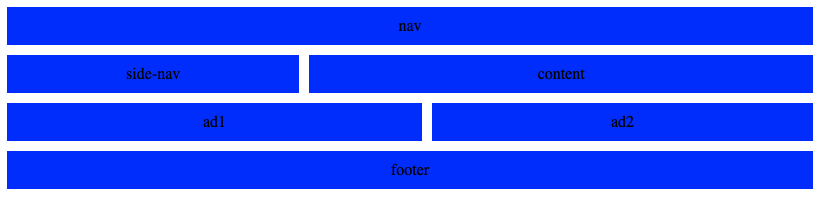
}

.container div {

background: blue;

padding: 10px;

}

Efekt?  
  
  
  
Ok, ale co tu się dzieje?

* Przede wszystkim, na początku zrobiliśmy sobie container w naszym pliku HTML.
* Wrzuciliśmy do niego divy odpowiadające sekcjom, które chcielibyśmy wyświetlić.
* W CSS zdefiniowaliśmy klasy tych elementów, jedyne co w nich dodajemy do parametr grid-area, którego zadaniem jest przypisanie nazwy tej części naszego grida.
* Opisaliśmy klasę containera. Ustawiamy oczywiście display na grid i pozwalamy kolumnom oraz wierszom dostosowywać swój rozmiar odpowiednio do zawartości - auto.
* Tutaj zaczynają się czary :) W grid-template-areas możemy użyć zdefiniowanych przez nasz wcześniej grid-area. Wpisujemy je w jednym wierszu tyle razy ile chcemy żeby proporcjonalnie zajęły miejsca. Musimy pamiętać, że każdy wiersz musi mieć tyle samo kolumn! Czyli jeżeli nav ma zająć całą szerokość, a mamy 6 kolumn, to trzeba go wpisać 6 razy.
* Sprawiliśmy dla czytelności, że tekst się wyśrodkowuje.
* Dodaliśmy wszystkim div'om wewnątrz containera niebieskie tło oraz padding - w celach estetycznych i żeby się łatwiej wyróżniały dla przykładu.

**Podsumowanie**

Flex i Grid to potężne narzędzia, które dają nam olbrzymie możliwości i przyspieszają zdecydowanie pracę nad stronami i aplikacjami.

Pamiętajcie jednak o najważniejszej zasadzie - nim przejdziecie dalej, **spróbujcie napisać swoje małe strony przy ich użyciu. Nie kopiując i wklejając wartości, tylko robiąc to samemu** :)

Podobnie jak na końcu poprzedniego wpisu, gorąco zachęcamy do "wgryzienia" się bardziej w temat przy pomocy tutoriala grupy [CSS-Tricks](https://css-tricks.com/snippets/css/complete-guide-grid/), którzy przedstawiają wszystkie dostępne parametry gridu oraz wizualizują ich zachowanie.

**Zadanie domowe - zrób, nim przejdziesz dalej!**

Warto byłoby przećwiczyć tego Grida i zrozumieć na własnej skórze jak działa, dlatego Twoimi zadaniami będzie:

**Zrób siatkę, która będzie składać się z 4 wierszy po 6 kolumn (4x6)**

Niech każdy element ma szare tło oraz jest oddzielony od pozostałych o 15px.

**Stwórz grid, w którym:**

* obszar (area) *header* zajmie całą szerokość i 1/3 wysokości,
* obszar *sidebar* zajmie 50% szerokości i pozostałe 2/3 wysokości, po lewej stronie,
* po prawej stronie obszar *main* oraz *footer* zajmą pozostałe 50% szerokości i oba po 1/3 wysokości, jeden pod drugim.

Tekstowa interpretacja zadania drugiego:

"header header"

"sidebar main "

"sidebar footer";