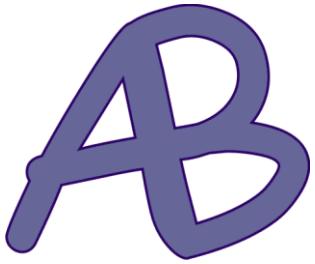


Media Query

Media Query

Zapytania @media (ang. media queries) to mechanizm w CSS, który umożliwia stosowanie różnych reguł stylów w zależności od cech urządzenia, na którym wyświetlana jest strona.

Dzięki nim projektant może dostosować układ, wygląd lub funkcjonalność interfejsu w zależności od:

- szerokości i wysokości ekranu (width, height),
- typów urządzeń (screen, print, tv),
- orientacji ekranu (orientation: portrait / landscape),
- rozdzielczości (resolution).

```
@media (warunek) {
  /* reguły CSS obowiązujące tylko, gdy warunek jest spełniony */
}
```

Media Query

Reguła @media określa docelowe typy nośników (oddzielone przecinkami) zestawu instrukcji (rozdzielanych nawiasami klamrowymi).

Nieprawidłowe instrukcje muszą być ignorowane.

Konstrukcja @media zezwala na reguły arkusza stylów dla różnych nośników w tym samym arkuszu stylów:

```
@media print {
  /* styl dla wydruku */
}
@media screen {
  /* styl dla ekranu */
}
@media screen, print {
    /* styl dla ekranu i wydruku */
}
```

Media Query

all - Nadaje się do wszystkich urządzeń.

braille - Przeznaczony do dotykowych urządzeń sprzężenia zwrotnego w alfabecie Braille'a.

embossed - Przeznaczony do stronicowanych drukarek brajlowskich

handheld - Przeznaczony do urządzeń przenośnych

print - Przeznaczony do materiałów stronicowanych i dokumentów wyświetlanych na ekranie w trybie podglądu wydruku.

projection - Przeznaczony do projekcji prezentacji, na przykład projektorów.

screen - Przeznaczony przede wszystkim do kolorowych ekranów komputerowych.

speech - Przeznaczony do syntezatorów mowy. Uwaga: CSS2 miał podobny typ nośnika o nazwie "dźwiękowy" w tym celu.

Media Query

Obecnie istnieją dwa sposoby określania zależności między mediami dla arkuszy stylów:

- Określ nośnik docelowy z arkusza stylów z @media lub @import regułami

```
@import url("fancyfonts.css") screen;
@media print {
  /* style sheet for print goes here */
}
```

- Określ nośnik docelowy w języku dokumentu. Na przykład w HTML 4 atrybut "media" w elemencie LINK określa nośnik docelowy zewnętrznego arkusza stylów:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<HTML>
  <HEAD>
    <TITLE>Link to a target medium</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" MEDIA="print, handheld" HREF="foo.css">
  </HEAD>
  <BODY>
    <P>The body...
  </BODY>
</HTML>
```

Media Query

Zapytanie mediów składa się ze standardowego typu lub grupy medium, po którym następuje słowo kluczowe "and" (ang. i), a następnie w nawiasie cecha medium (ang. media feature), określająca wymagane możliwości urządzenia w ramach podanego na początku mediów ogólnego.

```
@media screen and (max-width: 800px) and (min-width: 400px){
  /* jeśli szerokość jest mniejsza od 800px i większa  od 400px */
  body{ background-color: lightblue; }
}
```

Media Query

Właściwości dla reguły @media.	
width	określenie wartości szerokości okna przeglądarki internetowej
height	określenie wartości wysokości okna przeglądarki internetowej
device-width	określenie wartości rozdzielczości ekranu urządzenia (szerokość)
device-height	określenie wartości rozdzielczości ekranu urządzenia (wysokość)
color	określenie liczby bitów na kolor lub określenie czy urządzenie posiada kolorowy ekran
color-index	określenie wartości głębi kolorów, które obsługuje dane urządzenie
aspect-ratio	określenie wartości proporcji szerokości do wysokości okna przeglądarki internetowej
device-aspect-ratio	określenie wartości proporcji szerokości do wysokości rozdzielczości ekranu urządzenia
grid	określenie urządzenia z ograniczonymi możliwościami wyświetlania
monochrome	określenie liczby bitów na piksel w urządzeniach monochromatycznych, jednokolorowych
orientation	określenie orientacji pionowej lub poziomej urządzenia
resolution	określenie wartości gęstości pikseli dla danego urządzenia
scan	określenie czy urządzenie posiada skanowanie obrazu progresywne czy międzyliniowe

Media Query

Operatory logiczne dla reguł **@media.**

and	operator **" i "**, służy do tworzenia bardziej precyzyjnych warunków w regule **@media**
„przecinek”	Operator „przecinek” oznacza **" lub "**, służy do tworzenia bardziej precyzyjnych warunków w regule **@media**
not	operator **" negacji "**, służy do tworzenia bardziej precyzyjnych warunków w regule **@media**
only	operator przeznaczony dla starszych przeglądarek internetowych

Media Query

Atrybuty przydatne do tworzenia warunków

Inne atrybuty, które warto znać to:

min-width	Warunki obowiązują w każdej przeglądarce, której szerokość przekracza wartość podaną w zapytaniu.
max-width	Reguły dotyczą każdej przeglądarki, której szerokość jest poniżej wartości podanej w zapytaniu.
min-height	Reguły będą obowiązywać w każdej przeglądarce, której wysokość przekroczy wartość podaną w zapytaniu.
max-height	Warunki obowiązują w każdej przeglądarce, której wysokość jest poniżej wartości podanej w zapytaniu.
orientation=portrait	Reguły będą w każdej przeglądarce, której wysokość jest równa szerokości lub większa.
orientation=landscape	Reguły obowiązują w każdej przeglądarce, której szerokość jest większa niż wysokość.

viewport

Samo dodanie warunków @media nie sprawi, że nasza strona będzie się dobrze skalować na urządzeniach mobilnych. Trzeba na stronie pomiędzy znacznikami <head></head> dodać odpowiednią dyrektywę:

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

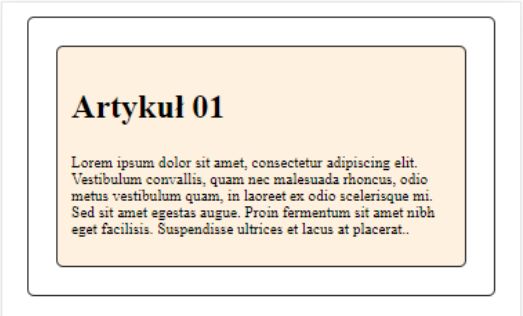
Służy ona kontrolowania widocznego obszaru (viewportu) w przeglądarkach mobilnych.

- name="viewport" – informuje przeglądarkę, że modyfikujemy ustawienia widocznego obszaru,
- content="width=device-width" – ustawia szerokość viewportu równą szerokości ekranu urządzenia,
- initial-scale=1.0 - ustawia początkowy poziom powiększenia strony i zapewnia naturalne odwzorowanie pikseli (1 CSS px ≈ 1 fizyczny px na urządzeniu).

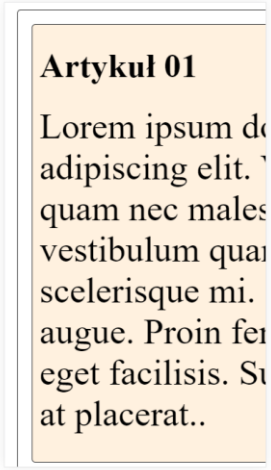
viewport

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flex</title>
    <link rel="stylesheet" type="text/css" href="styl01.css" />
  </head>
  <body>
    <section>
      <article>
        <h2>Artykuł 01</h2>
        <p>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
          convallis, quam nec malesuada rhoncus, odio metus vestibulum quam, in
          laoreet ex odio scelerisque mi. Sed sit amet egestas augue. Proin
          fermentum sit amet nibh eget facilisis. Suspendisse ultrices et lacus
          at placerat..
        </p>
      </article>
    </section>
  </body>
</html>
```

```css
section,
article {
  margin: 10px;
  padding: 10px;
  border: 1px solid rgb(47, 47, 47);
  border-radius: 5px;
}
p {
  font-size: 2vw;
}
article {
  background-color: rgb(255, 241, 223);
}
```



Bez tagu viewport



viewport

Domyślne zachowanie bez tej dyrektywy

W przypadku braku tego meta tagu, wiele przeglądarek mobilnych symuluje szerokość ekranu około 980px lub więcej, skalując całą stronę w dół.
Powoduje to, że:

- teksty są bardzo małe,
- układ nie pasuje do ekranu,
- trzeba ręcznie powiększać stronę (pinch-to-zoom).

Efekt zastosowania width=device-width:

- przeglądarka ustawia viewport na rzeczywistą szerokość ekranu urządzenia,
- dzięki temu media queries (@media (min-width: 600px)) działają poprawnie,
- układ CSS (np. Flexbox, Grid) zachowuje się zgodnie z projektem responsywnym.

viewport

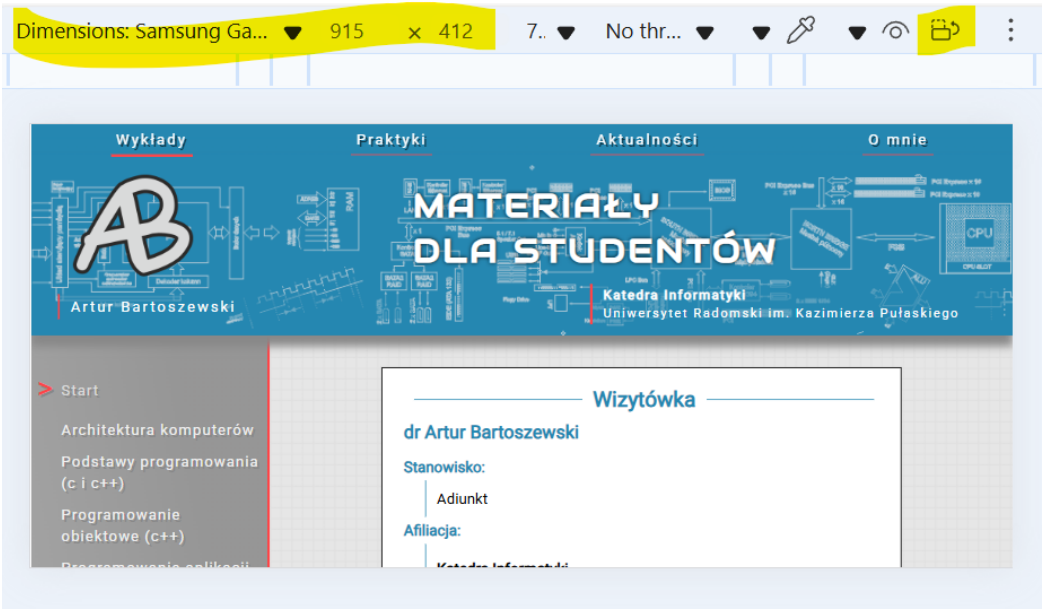
Viewport to tzw. obszar renderowania treści w wirtualnym oknie przeglądarki wyświetlanym na ekranie. Mówiąc najprościej, to obszar przeglądarki, w którym widzimy stronę www.

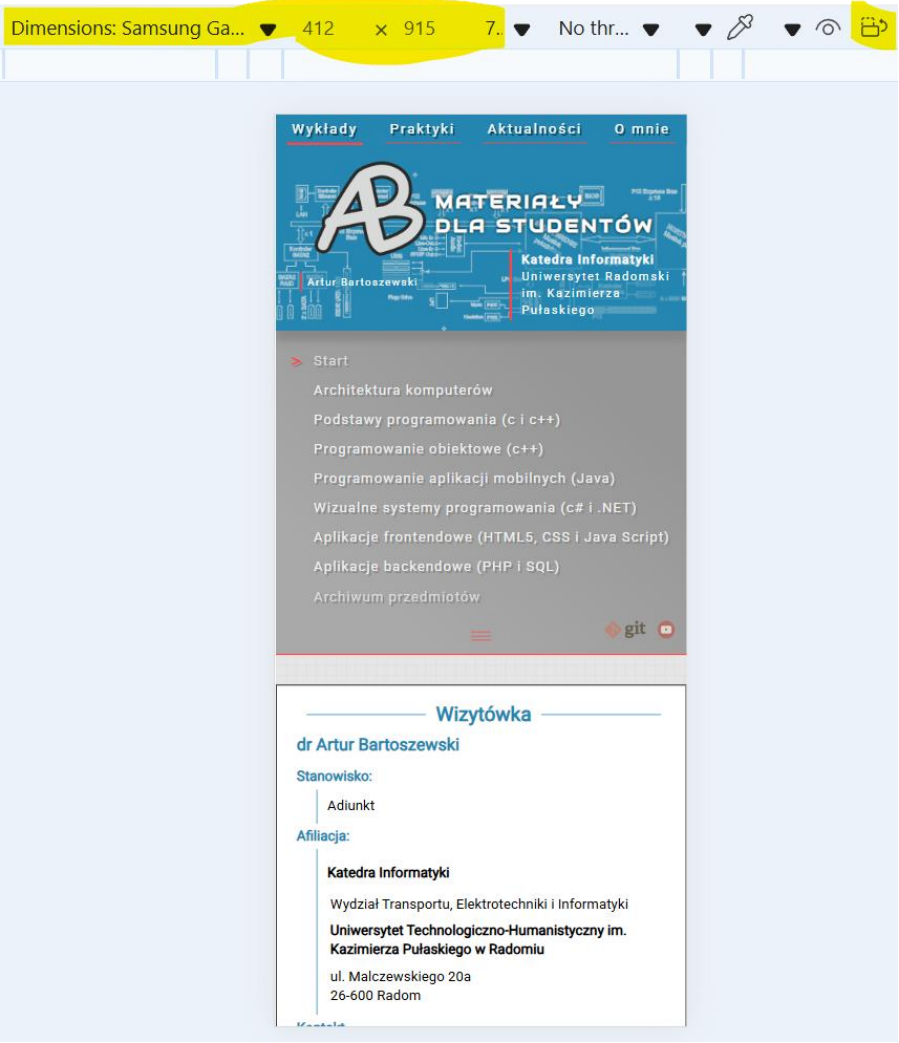
Viewport a rozdzielczość ekranu:

- Rozdzielczość ekranu (ang. screen resolution) – liczba fizycznych pikseli na ekranie (np. 1080x1920).
- Viewport – logiczna przestrzeń, w której przeglądarka układa i rysuje stronę, nie zawsze równa fizycznym pikselom.

viewport

Na smartfonach viewport jest często skalowany, aby strona stworzona na desktop wyglądała „jako tako” – przeglądarka udaje, że ekran ma np. 915px wysokości (zależnie od telefonu)





viewport

Jednostki viewport CSS

Aby uzyskać dynamicznie zmieniające się rozmiary elementów na stronie względem okna przeglądarki, należy ustawić im rozmiar za pomocą jednostek tzw. Viewport Units, które reprezentują procent aktualnych wymiarów obszaru roboczego ekranu:

vw – (viewport width) procentowa szerokość – względem szerokości okna przeglądarki;

vh – (viewport height) procentowa wysokość – względem wysokości okna przeglądarki;

vmin – (viewport minimum) mniejsza z wartości vw lub vh;

vmax – (viewport maximum) większa z wartości vw lub vh.

viewport

Jednostki viewport CSS

Aby uzyskać dynamicznie zmieniające się rozmiary elementów na stronie względem okna przeglądarki, należy ustawić im rozmiar za pomocą jednostek tzw. Viewport Units, które reprezentują procent aktualnych wymiarów obszaru roboczego ekranu:

vw – (viewport width) procentowa szerokość – względem szerokości okna przeglądarki;

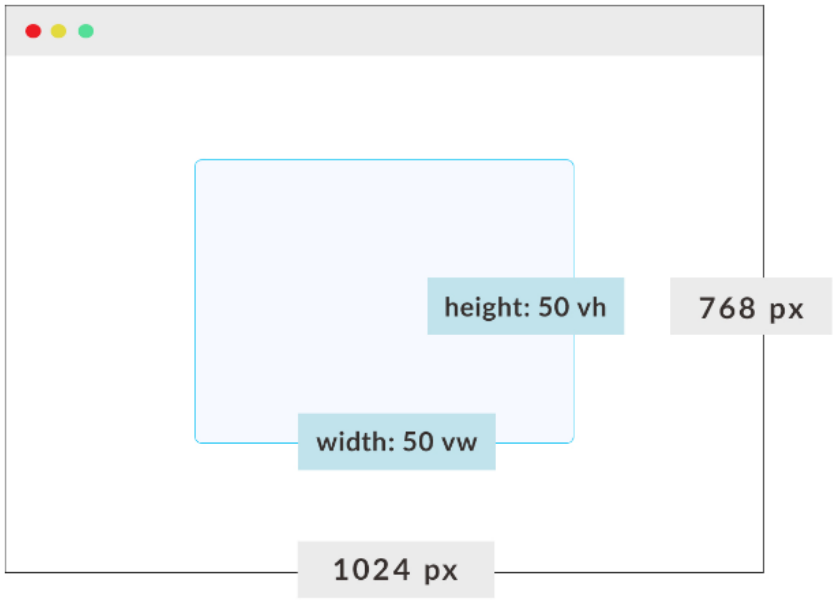
vh – (viewport height) procentowa wysokość – względem wysokości okna przeglądarki;

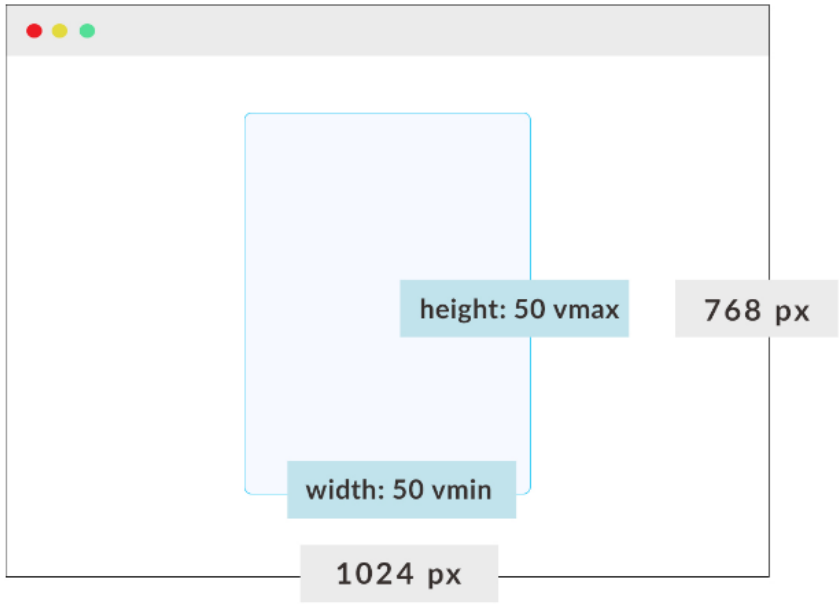
vmin – (viewport minimum) mniejsza z wartości vw lub vh;

vmax – (viewport maximum) większa z wartości vw lub vh.

viewport

Źródło: https://ui2web.com/frontend/jednostki-viewport-na-czym-polegaja/





width: 50vw – element o wysokości połowy wysokości aktualnego okna przeglądarki
height: 50vh – element o wysokości połowy wysokości aktualnego okna przeglądarki

width: 50vmin – element o szerokości połowy z aktualnie większej wartości – wysokości lub szerokości

height: 50vmax – element o wysokości połowy z aktualnie mniejszej wartości – wysokości lub szerokości

Funkcja CSS calc()

Funkcja calc() przy zastosowaniu jednostek viewport i stałych jednostek zapobiega niekontrolowanemu zmniejszaniu rozmiarów elementu. Dzięki niej możemy ustanowić podstawową wartość tekstu na 14px i dodać do niej 2 jednostki vw.

```
.main-text {
  font-size: calc(14px + 2vw);
}
```

Aby zapobiec znacznemu powiększeniu tekstu na dużych ekranach, musimy zastosować standardowe Media Queries:

```
@media (min-width: 1600px) {
 .main-text {
   font-size:40px;
 }
}
```

Funkcja calc( )

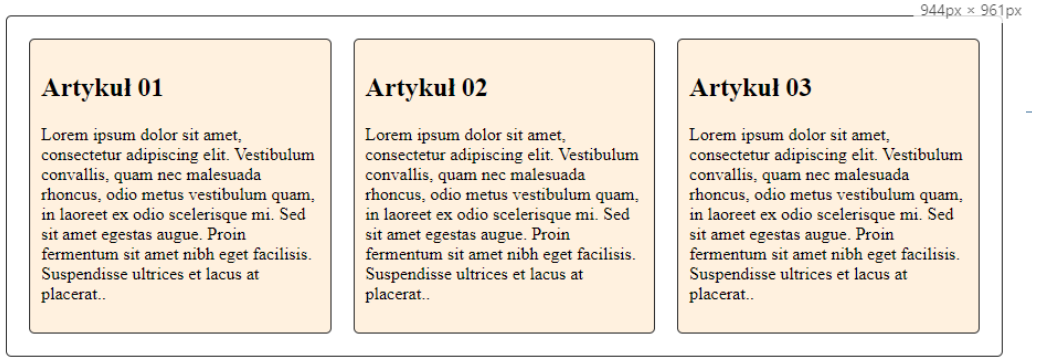
CSS ma specjalną funkcję do wykonywania podstawowej matematyki.

Przykłady:

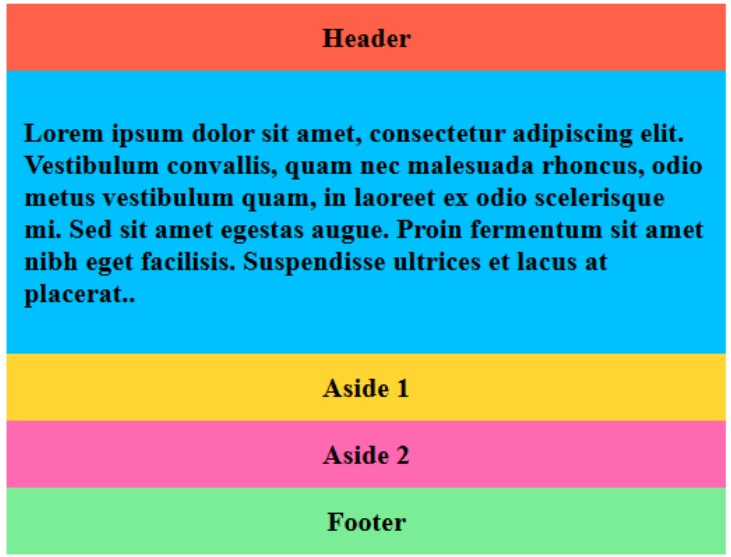
```
font-size: calc(3vw + 2px);
width: calc(100% - 20px);
height: calc(100vh - 20px);
padding: calc(1vw + 5px);
border-radius: 15px calc(15px / 3) 4px 2px;
```

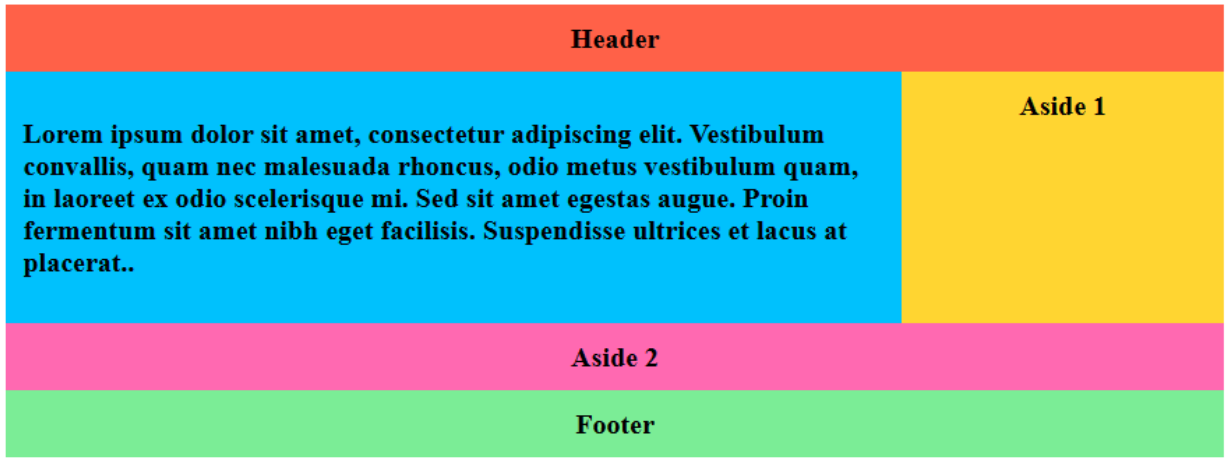
Przykład

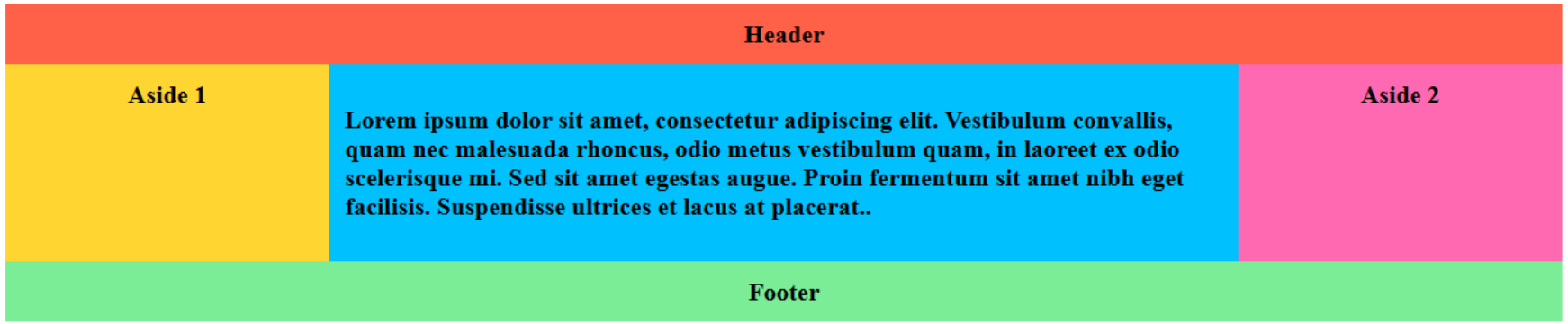
```
section,
article {
  margin: 10px;
  padding: 10px;
  border: 1px solid rgb(47, 47, 47);
  border-radius: 5px;
}

article {
  background-color: rgb(255, 241, 223);
}

body {
  display: flex;
  flex-wrap: wrap;
}
section {
  display: flex;
  flex-wrap: wrap;
}

@media screen and  (min-width: 800px){

  article {
    max-width: calc(33vw - 60px);
  }
}
```



Przykład 2 – Układ responsywny FLEX





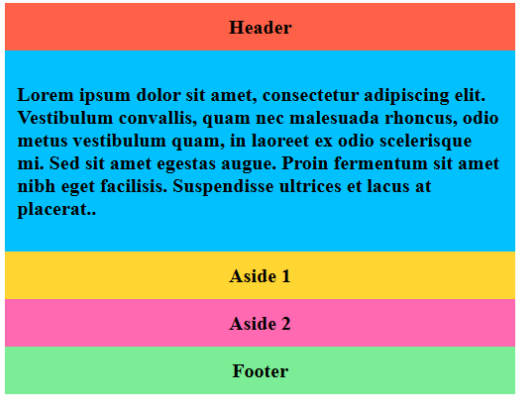


Przykład 2 – Układ responsywny FLEX

```
<body>
  <main>
    <header class="header">Header</header>
    <aside class="aside aside-1">Aside 1</aside>
    <article class="article">
      <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
        convallis, quam nec malesuada rhoncus, odio metus vestibulum quam, in
        laoreet ex odio scelerisque mi. Sed sit amet egestas augue. Proin
        fermentum sit amet nibh eget facilisis. Suspendisse ultrices et lacus
        at placerat.
      </p>
    </article>
    <aside class="aside aside-2">Aside 2</aside>
    <footer class="footer">Footer</footer>
  </main>
</body>
</html>
```

Przykład 2 – Układ responsywny FLEX

Najczęściej układem domyślnym jest układ dla najwyższego ekranu aktywny np.. na smartfonach.



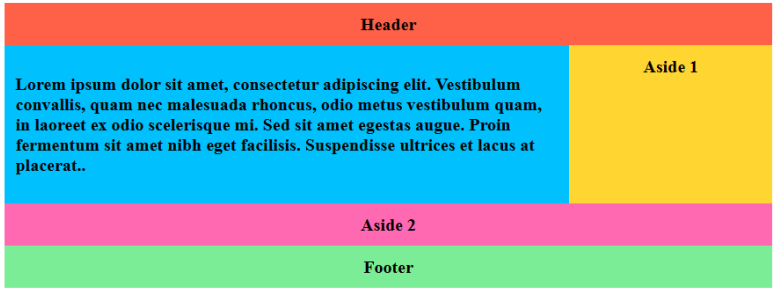

```css
/* Ustawienia dla całego dokumentu */
body {
  padding: 2em; /* Margines wewnętrzny dookoła zawartości */
}

/* Ustawienia głównego kontenera */
main {
  display: flex;              /* Włączenie modelu flexbox */
  flex-flow: row wrap;        /* Układ wierszowy z możliwością zawijania */
  font-weight: bold;          /* Pogrubiona czcionka dla wszystkich elementów */
  text-align: center;         /* Wyśrodkowanie tekstu */
}

/* Domyślne ustawienia wszystkich bezpośrednich dzieci <main> */
main > * {
  padding: 10px;              /* Wewnętrzny margines */
  flex: 1 1 100%;             /* Domyślnie każdy element zajmuje 100% szerokości */
}
```

Przykład 2 – Układ responsywny FLEX

```
/* Styl nagłówka */
.header {
  background: tomato;         /* Kolor tła */
  order: 10;                  /* Kolejność wyświetlania w układzie flex
*/
}

/* Styla stopki */
.footer {
  background: lightgreen;
  order: 50;
}
```

```
/* Styl artykułu */
.article {
  text-align: left;           /* Wyrównanie tekstu do lewej */
  background: deepskyblue;
  order: 20;
}

/* Styl pierwszego panelu bocznego */
.aside-1 {
  background: gold;
  order: 30;
}

/* Styl drugiego panelu bocznego */
.aside-2 {
  background: hotpink;
  order: 40;
}
```

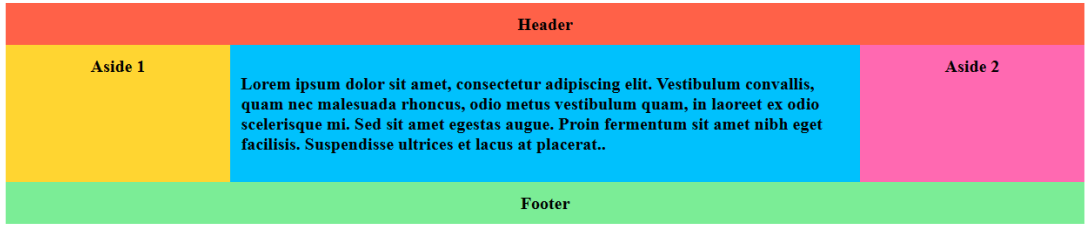
Przykład 2 – Układ responsywny FLEX




```
/* Media query dla szerokości minimum 600px */
@media all and (min-width: 600px) {
  .aside-1 {
    order: 25;                  /* Zmiana kolejności */
    flex: 1;                    /* Zajmuje 1 część przestrzeni */
  }

  .aside-2 {
    flex: 1 1 100%;             /* Zajmuje całą szerokość wiersza */
  }

  .article {
    flex: 3;                    /* Zajmuje 3 razy więcej miejsca niż aside-1
*/
  }
}
```

Użycie order pozwala sterować kolejnością renderowania elementów niezależnie od ich kolejności w HTML.

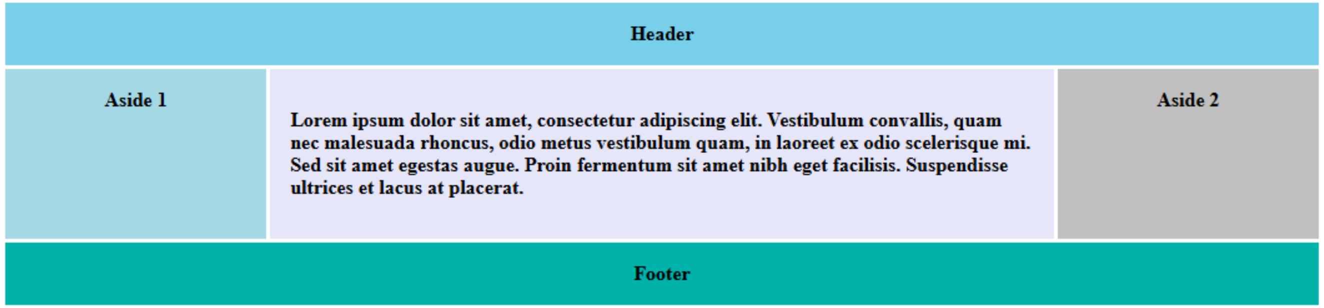
Przykład 2 – Układ responsywny FLEX

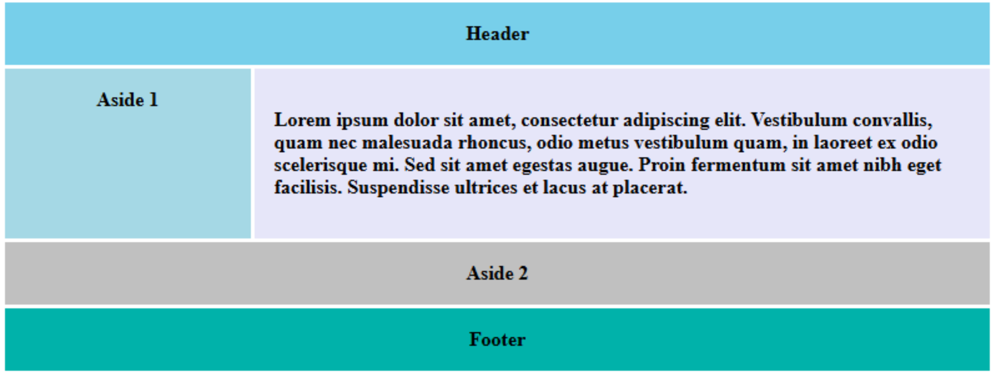


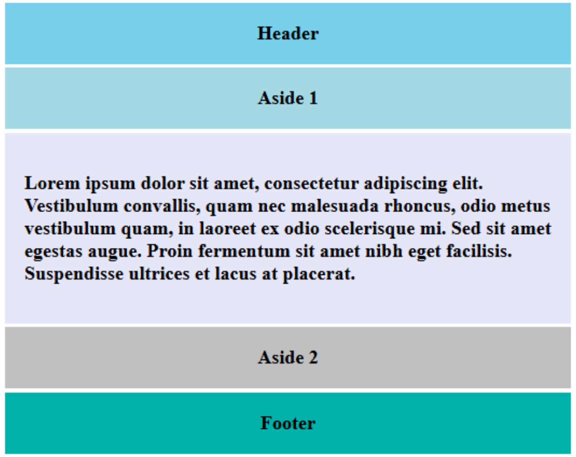

```
/* Media query dla szerokości minimum 1000px */
@media all and (min-width: 1000px) {
  .aside {
    flex: 1;                    /* Oba panele boczne mają równą szerokość */
  }

  .article {
    flex: 3;                    /* Artykuł nadal zajmuje 3 części */
  }

  .aside-1 {
    order: 20;                  /* Przesunięcie aside-1 przed artykuł */
  }
}
```

Przykład 3 - Układ responsywny GRID

Tym razem spróbujemy uzyskać analogiczny efekt przy użyciu systemu GRID







Przykład 3 - Układ responsywny GRID

```
<body>
  <main>
    <header class="header">Header</header>
    <aside class="aside aside-1">Aside 1</aside>
    <article class="article">
      <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
        convallis, quam nec malesuada rhoncus, odio metus vestibulum quam, in
        laoreet ex odio scelerisque mi. Sed sit amet egestas augue. Proin
        fermentum sit amet nibh eget facilisis. Suspendisse ultrices et lacus
        at placerat.
      </p>
    </article>
    <aside class="aside aside-2">Aside 2</aside>
    <footer class="footer">Footer</footer>
  </main>
</body>
```

W pliku HTML dobra dodane zostały klasy jednoznacznie identyfikujące wszystkie elementy

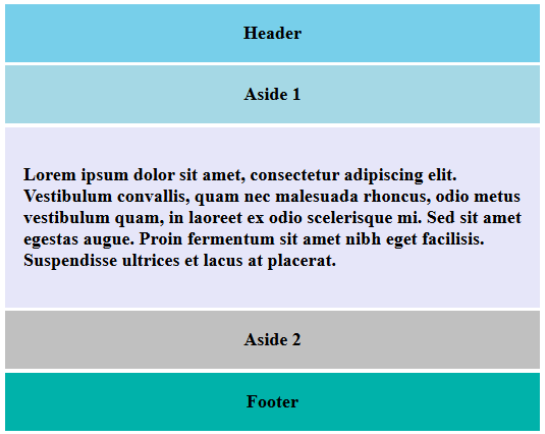
Przykład 3 - Układ responsywny GRID

```css
body {
  padding: 2em;
}

/* Stylizacja wszystkich elementów */
.header, .footer, .article, .aside-1, .aside-2 {
  padding: 1em; /* Wewnętrzny margines */
  text-align: center; /* Wyśrodkowanie tekstu */
}
```

```css
/* Przypisanie elementów do obszarów siatki */
.header {
  background: skyblue;
  grid-area: header;
}

.footer {
  background: lightseagreen;
  grid-area: footer;
}

.article {
  background: lavender;
  grid-area: article;
  text-align: left;
}

.aside-1 {
  background: lightblue;
  grid-area: aside1;
}

.aside-2 {
  background: silver;
  grid-area: aside2;
}
```

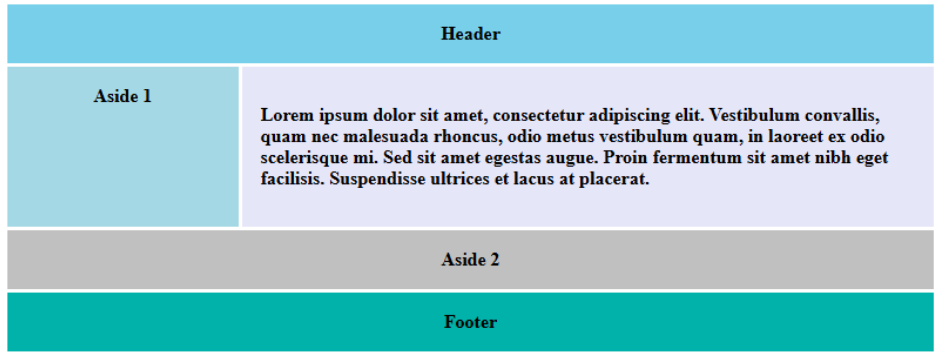
Przykład 3 - Układ responsywny GRID

```
main {
  display: grid;
  gap: 3px; /* Odstępy między elementami */
  grid-template-areas:
    "header"
    "aside1"
    "article"
    "aside2"
    "footer";
  grid-template-columns: 1fr; /* Jedna kolumna domyślnie */
 }
```



Przykład 3 - Układ responsywny GRID

```
/* Layout dla szerokości od 600px */
@media (min-width: 600px) {
  main {
    grid-template-columns: 1fr 3fr; /* 2 kolumny: aside i content */
    grid-template-areas:
      "header header"
      "aside1 article"
      "aside2 aside2"
      "footer footer";
  }
}
```



Przykład 3 - Układ responsywny GRID

```
/* Layout dla szerokości od 1000px */
@media (min-width: 1000px) {
  main {
    grid-template-columns: 1fr 3fr 1fr; /* 3 kolumny */
    grid-template-areas:
      "header header header"
      "aside1 article aside2"
      "footer footer footer";
  }
}
```