



# Media queries i viewport

# Czym są Media queries?

## Co to takiego?

Za pomocą reguły `@media` możemy decydować o tym, która część kodu CSS ma być interpretowana przez przeglądarkę na danym urządzeniu.

Dzięki temu możemy określić, jak treść – zależnie od urządzenia – będzie tam prezentowana.

W specyfikacji CSS2 przy użyciu reguły `@media` można było wybrać rodzaj medium:

- `screen`
- `print`

Specyfikacja CSS3 pozwoliła na sprecyzowanie kryteriów reguły `@media` przez zwiększenie zakresu rodzaju medium i dodanie właściwości takich jak szerokość.

# Rodzaje mediów

## Rodzaje media

- **all** – dla wszystkich urządzeń,
- **screen** – dla kolorowych ekranów komputerowych lub mobilnych,
- **tv** – dla telewizorów,
- **print** – dla wydruku,
- **projection** – dla prezentacji/projektorów,
- **speech** – dla syntezatorów mowy,

- **braille** – dla dotykowych urządzeń obsługujących system Braille'a,
- **embossed** – dla drukarek obsługujących system Braille'a,
- **handheld** – dla prostych urządzeń komórkowych z prostym wyświetlaczem,
- **tty** – dla mediów ze stałą liczbą znaków w jednym wierszu.

# Czym jest viewport?

## Obszar widoczny

**viewport** to widoczny obszar ekranu, na jakim użytkownik wyświetla stronę.

Na jego podstawie powinny być sprawdzane warunki szerokości i wysokości zadeklarowane w **media queries**.

Z reguły tylko część strony mieści się w **widocznym obszarze**, pozostałą można zobaczyć dzięki przewinięciu jej zawartości.

## Co jeśli nie ustawimy viewportu?

Jeśli obszar widoczny nie jest skonfigurowany, urządzenia mobilne renderują stronę według rozdzielczości zastępczej, która z reguły waha się między **800px** a **1024px**.

W wyniku tego strona zostanie przeskalowana według tej rozdzielczości tak, aby zapełniła ekran urządzenia. Zmusza to użytkownika do powiększenia strony, aby z niej skorzystać.

# Metatag viewport - ustawienie

## Wielkość strony

Aby strona mogła być responsywna i jej wielkość odpowiednio skalowalna do obszaru widocznego urządzenia, należy zdefiniować `viewport` za pomocą `metatagu`.

Taki tag powinien znaleźć się w sekcji `head` każdej strony.

```
<meta name="viewport">
```

## Składnia

```
<meta name="viewport" content=" " >
```

5

## Składnia - przykład

```
<meta name="viewport" content="initial-scale=1">
```

# Metatag viewport - ustawienie

## Wielkość strony

Aby strona mogła być responsywna i jej wielkość odpowiednio skalowalna do obszaru widocznego urządzenia, należy zdefiniować `viewport` za pomocą `metatagu`.

Taki tag powinien znaleźć się w sekcji `head` każdej strony.

```
<meta name="viewport">
```

## Składnia

```
<meta name="viewport" content=" " >
```

Tu wpisujemy wartości.

## Składnia - przykład

```
<meta name="viewport" content="initial-scale=1">
```

# Metatag viewport

## Wartości

Metatag `viewport` może przyjąć następujące wartości, które wpisujemy w atrybucie `content`:

- `width i height` – pozwala zdefiniować szerokość i wysokość obszaru widocznego,
- `initial-scale` określa domyślne powiększenie (zoom) strony,
- `minimum-scale, maximum-scale` – określa wielkość minimalnej (0) i maksymalnej (10) wielkości strony

## Przykład:

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

# Metatag viewport

## Brak metatagu viewport

Na stronie obok nie ma określonego viewportu. Strona otwarta była na urządzeniu iPhone 7 Plus o rozdzielczości ekranu (podanej przez producenta):

- Szerokość: **414px**
- Wysokość: **736px**

Wymiary urządzenia odczytane z obiektu `window` w JavaScript wskazuje na następujące wymiary:

- Szerokość: **980px**
- Wysokość: **1468px**



# Metatag viewport

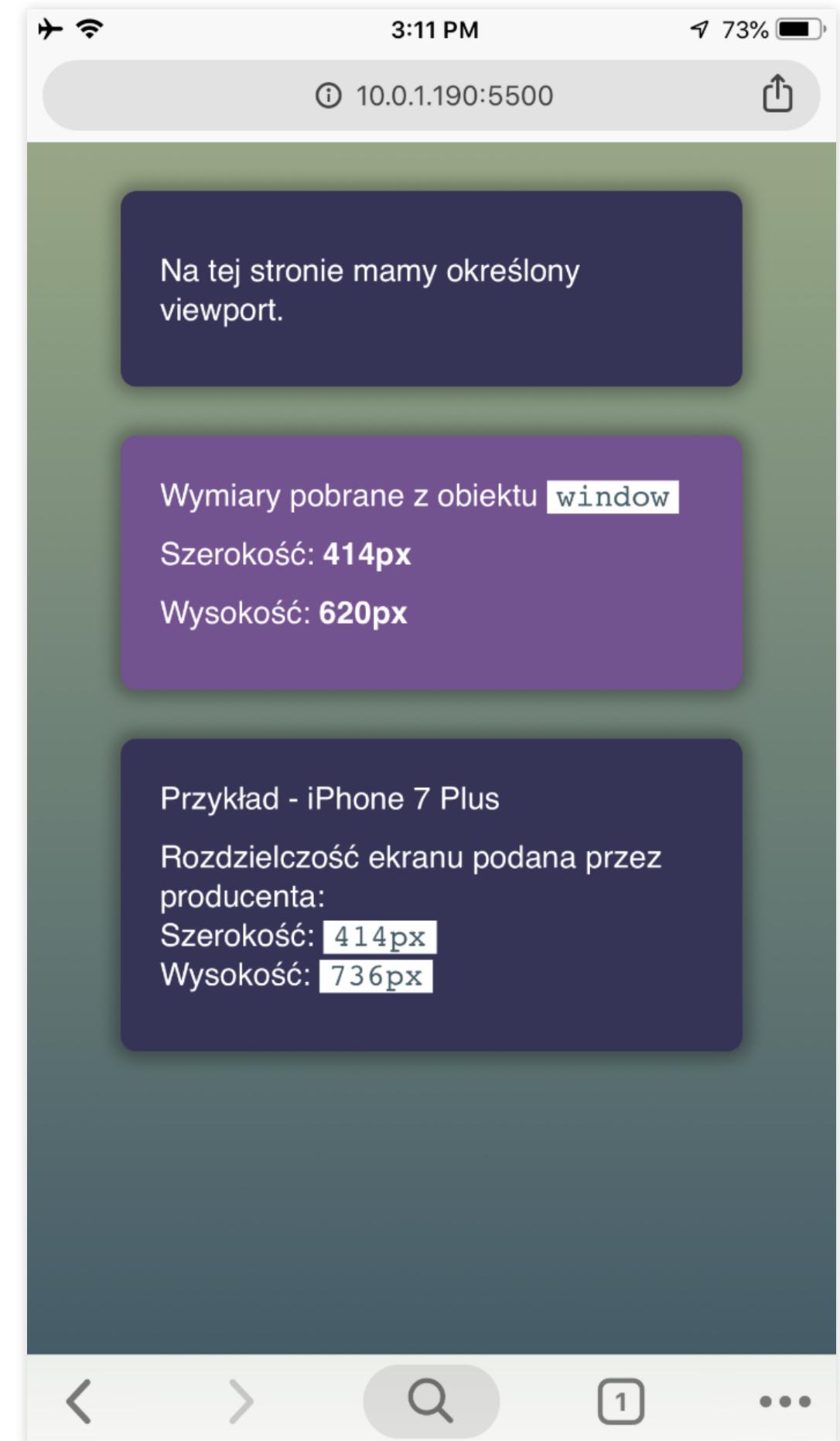
## Efekt użycia metatagu viewport

Na stronie obok został określony metatag viewportu.

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

Wymiary urządzenia odczytane z obiektu `window` są już podobne do wymiarów podanych przez producenta (mamy mniejszą wartość wysokości ze względu na górną i dolną belkę którą zajmuje przeglądarka):

- Szerokość: `414px`
- Wysokość: `620px`



# Budowanie media queries

## Kilka słów o @media

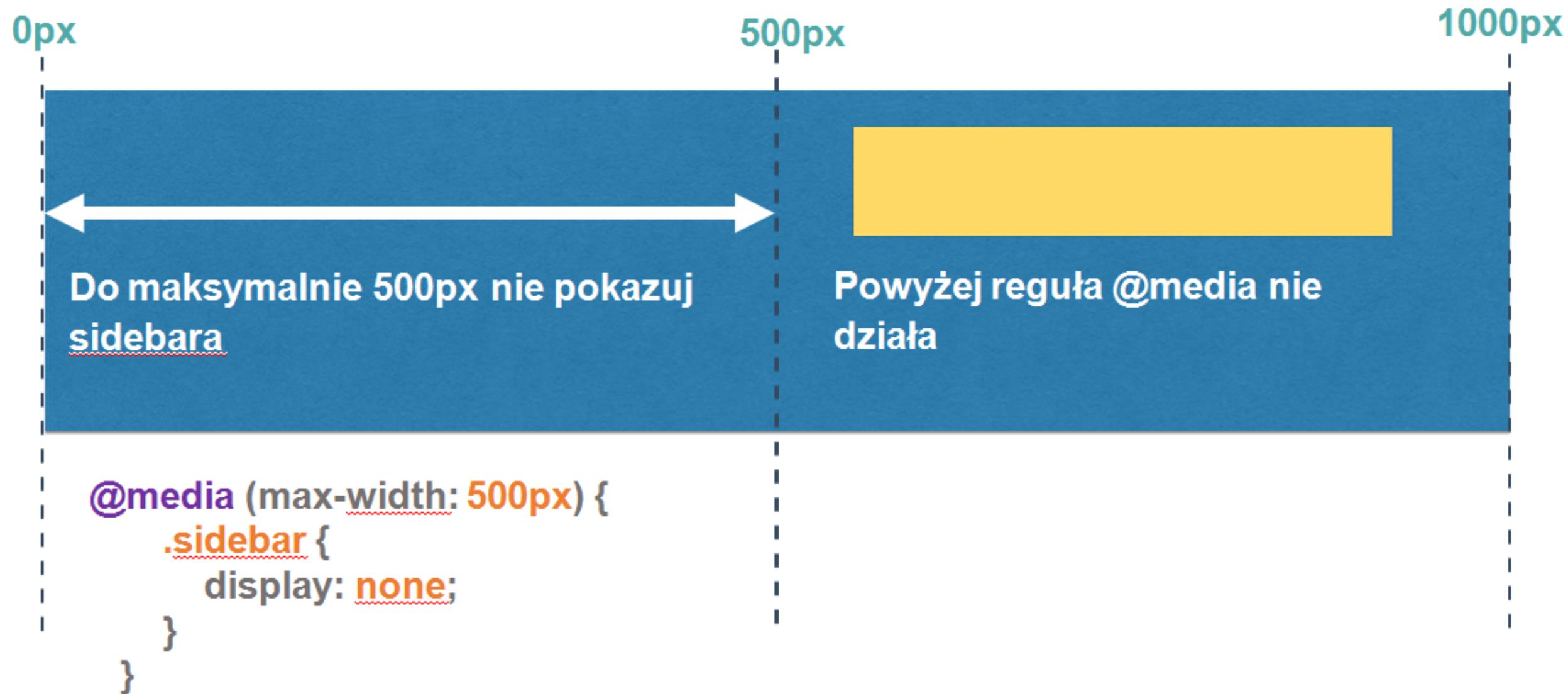
- Po wstawieniu odpowiedniego tagu `<meta>`, optymalną metodą jest wstawienie reguł `@media` w jednym pliku CSS.
- Efektywna reguła `@media` powinna składać się z określenia rodzaju medium i jego właściwości.
- Najbardziej przydatnymi właściwościami reguł `@media` są szerokość, orientacja i gęstość pikseli.

## Przykład

```
@media (max-width: 600px) {  
    .sidebar {  
        display: none;  
    }  
}
```

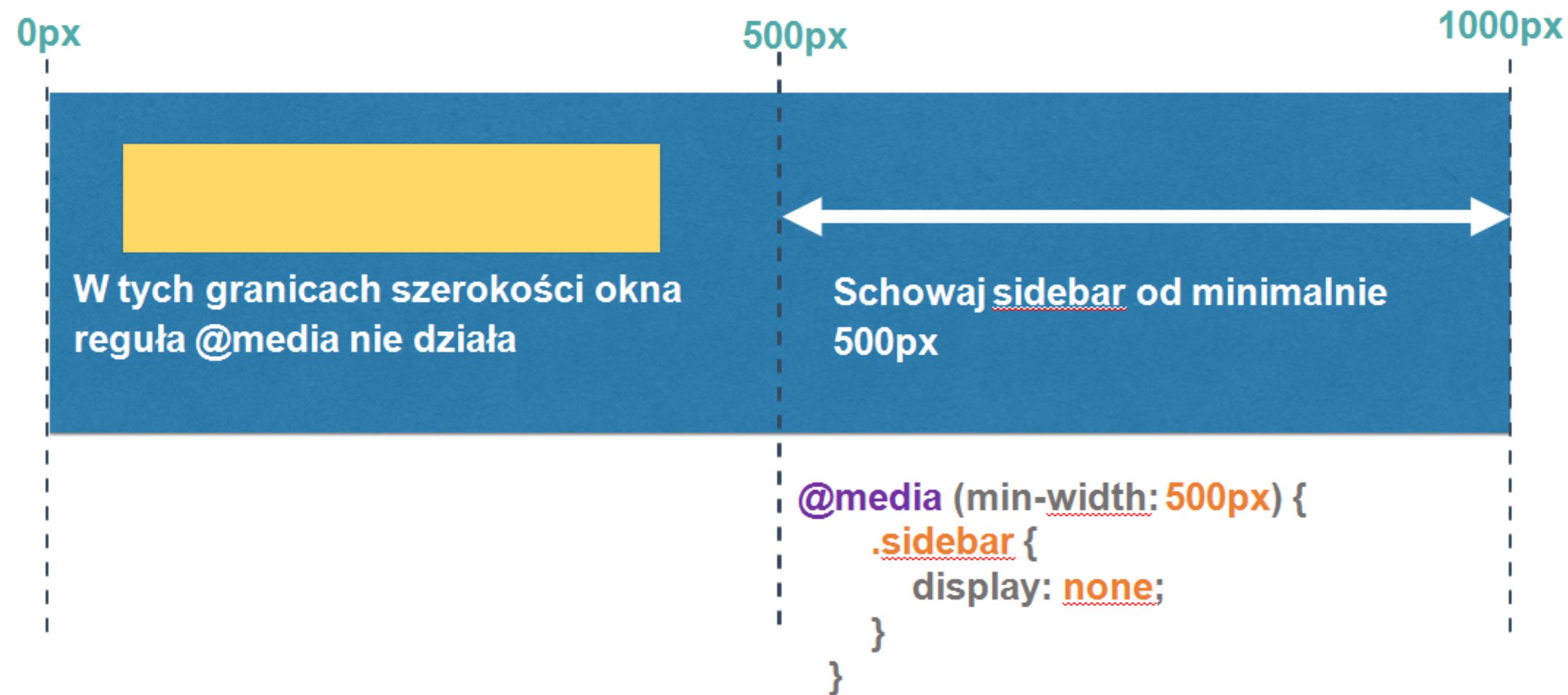
# Szerokość - max-width

Jak czytamy wartości max-width w media?



# Szerokość - min-width

Jak czytamy wartości min-width w media?

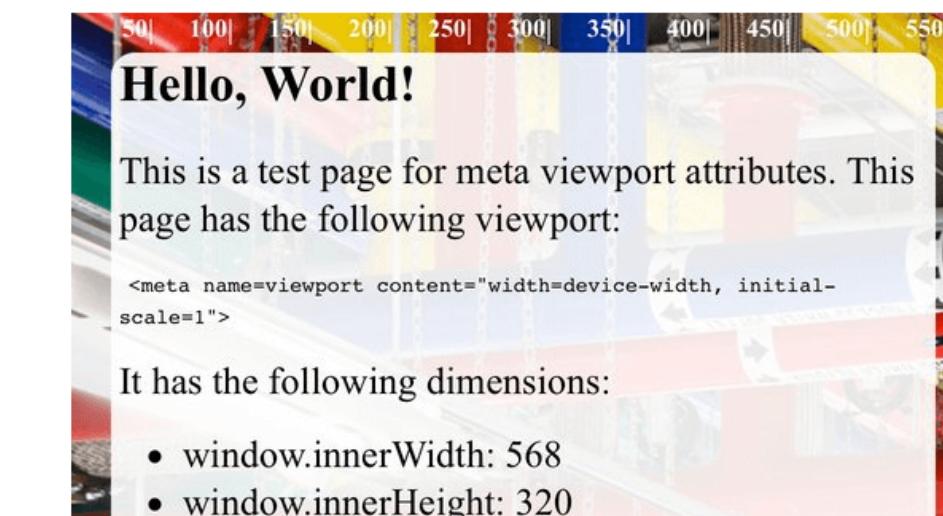


# Orientacja

## Orientacja pionowa



## Orientacja pozioma



# Gęstość pikseli a CSS

## Piksel fizyczny

Jeden piksel CSS może być reprezentowany przez wiele pikseli fizycznych urządzenia. Taką relację określa się przez wskaźnik **dppx** (dots per px unit).

Jeśli przyjmiemy wartość **1dppx**, to jeden piksel fizyczny odpowiada jednemu pikselowi CSS.

W takim przypadku obrazy o standardowej gęstości pikseli (1 do 1) często wyglądają fatalnie na urządzeniach z większą gęstością pikseli.

Dlatego dla wyświetlaczy z wyższą gęstością pikseli serwuje się obrazki o wyższej definicji.

Np. jeśli obraz ma wymiary CSS **200x200px**, to dla wyświetlacza z rozdzielczością **2dppx** powinien być serwowany obraz o rozdzielczości **400x400** pikseli, w czym pomogą nam **media queries**.

# Łączenie warunków media

## Media query czyli wyrażenie logiczne **if**

**Media queries** wykorzystują następujące operatory do łączenia warunków:

- **and** (x and y),
- **przecinek** (x or y),
- **not** (np. not x and y = not (x and y) ≠ (not x) and y).

## Operator **and**

Jeśli chcemy by **@media** spełniało kilka warunków **jednocześnie** używamy **and**.

```
@media (min-width: 700px) and  
(orientation: landscape) {  
}  
  
@media tv and (min-width: 700px) and  
(orientation: landscape) {  
}
```

# Operator AND

## Operator and

Jeśli chcemy by `media query` spełniało kilka warunków jednocześnie używamy `and`. Oba warunki muszą być spełnione.

```
@media (min-width: 700px) and (orientation: landscape) {  
    .sidebar {  
        display: none;  
    }  
}
```

# Operator OR

## Operator przecinek

Przecinek łączy warunki tak jak operator logiczny `or`.

Oznacza to, że jeśli którykolwiek z warunków zostanie spełniony, to dana część kodu CSS będzie interpretowana.

```
@media (min-width: 700px), screen and (orientation: landscape) {  
    .sidebar {  
        display: none;  
    }  
}
```

# Operator NOT

## Operator not

Jeśli chcemy, by `media query` nie było przeznaczone dla mediów spełniających dane warunki, używamy operatora `not`.

Operator ten neguje całe wyrażenie i nie może być zastosowany do poszczególnych warunków łączonych przez operator `and` lub `przecinek`.

```
@media not print {  
    .sidebar {  
        display: none;  
    }  
}
```

# Operator ONLY

## Kiedy używamy operatora only?

Używamy tego operatora, jeśli chcemy, by kod CSS nie był interpretowany przez starsze przeglądarki.

## Dlaczego?

Operator `only` występuje tutaj jako typ media, który nie istnieje. Starsze przeglądarki nie będą interpretowały zatem dalszej części warunku a tym samym kodu **CSS**.

```
@media only screen and (min-width: 600px) {  
}
```

# Właściwości mediów

- **min-width/min-height i max-width/max-height** - określenie zakresu wartości szerokości/wysokości obszaru widocznego.
- **min-device-width/min-device-height i max-device-width/max-device-height** - określenie zakresu wartości szerokości/wysokości urządzenia.

```
@media screen and (min-width: 500px)  
and (max-height: 800px) { ... }
```

```
@media screen and (min-device-height: 480px)  
and (max-device-width: 640px) { ... }
```

# Właściwości mediów

- **orientation** – określenie orientacji pionowej lub poziomej urządzenia,
- **resolution** – określenie wartości gęstości pikseli dla danego urządzenia.

```
@media all and (orientation: portrait) { ... }
```

```
@media print and (min-resolution: 300dpi) { ... }
```

```
@media screen and (min-resolution: 2dppx) { ... }
```

# Jednostki relatywne, responsywność i viewport

- RWD zakłada użycie jednostek relatywnych.
- Ze względu na dużą fragmentację rozmiarów ekranów, dobranie jednostek sztywnych (`px`)
  - tak by strona wyświetlała się prawidłowo na wszystkich ekranach – jest bardzo trudne.  
Dlatego lepszym rozwiązaniem jest używanie jednostek relatywnych takich jak:
    - `procenty`,
    - `emy`,
    - `remy`.
- CSS3 pozwala na używanie jednostek relatywnych w odniesieniu do wielkości `viewport`:
  - `vh` (viewport height),
  - `vw` (viewport width),
  - `vmin` i `vmax`.

Jedna jednostka viewport to 1% jego szerokości lub wysokości.



# Media queries w Sass

# Media query ze zmienią

## Przykład

```
$tablet: "(min-width: 768px) and (max-width: 1023px)";  
$desktop: "(min-width: 1024px)";  
  
p {  
    font-size: 16px;  
    @media #{$tablet} {  
        font-size: 18px;  
    }  
    @media #{$desktop} {  
        font-size: 20px;  
    }  
}
```

# Media query jako mixin

## Składnia SCSS

```
$tablet-width: 768px;  
@mixin tablet {  
  @media (min-width: #{$tablet-width}) {  
    @content;  
  }  
}  
.main-section {  
  margin: 0 auto;  
  @include tablet {  
    padding: 6px 12px;  
    width: 900px;  
  }  
}
```

## Składnia CSS

```
.main-section {  
  margin: 0 auto;  
}  
@media (min-width:768px) {  
  .main-section {  
    padding: 6px 12px;  
    width: 900px;  
  }  
}
```

# Media query jako mixin

## Składnia SCSS

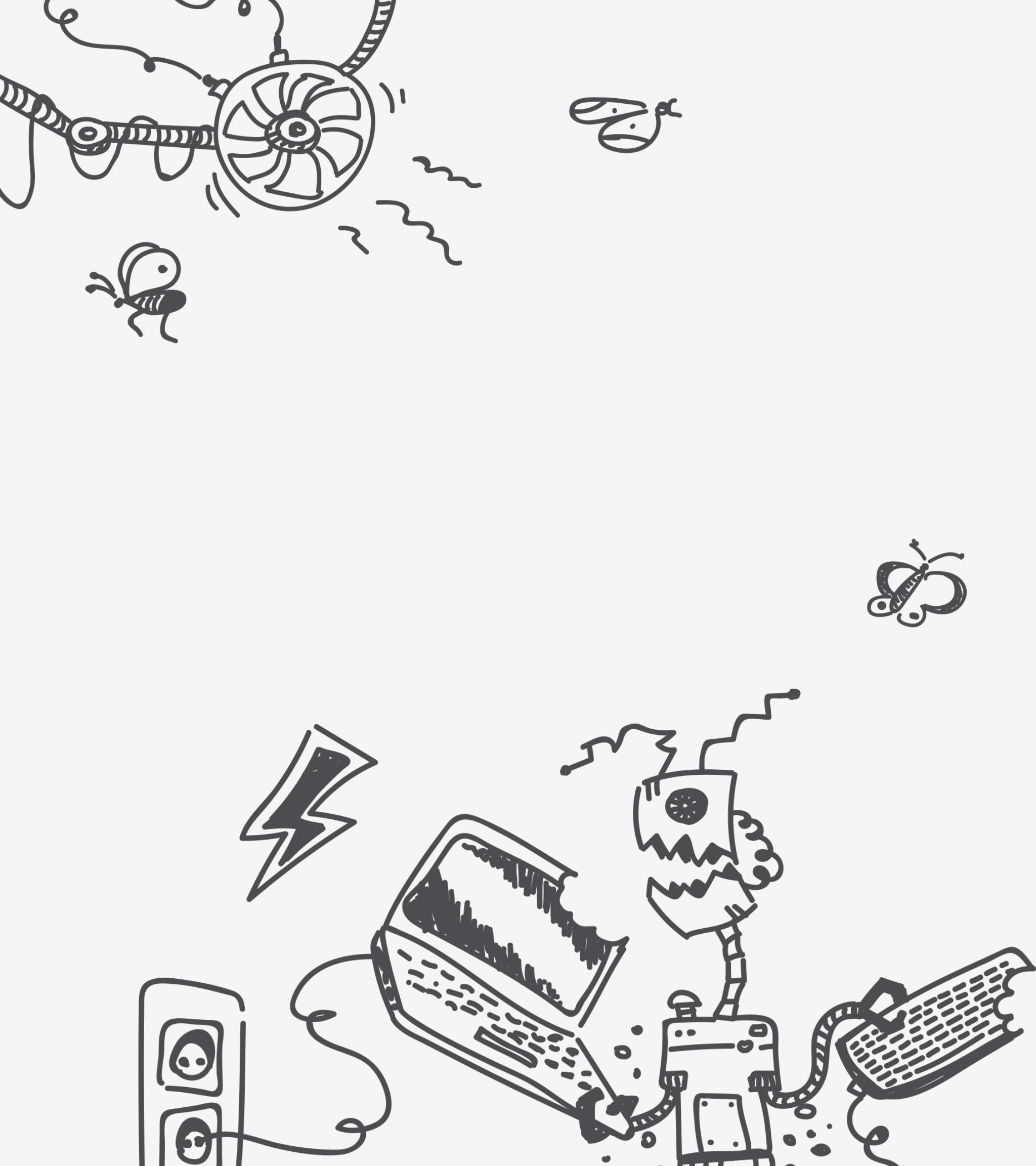
```
$tablet-width: 768px;  
@mixin tablet {  
  @media (min-width: #{$tablet-width}) {  
    @content;  
  }  
}  
.main-section {  
  margin: 0 auto;  
  @include tablet {  
    padding: 6px 12px;  
    width: 900px;  
  }  
}
```

## Składnia CSS

```
.main-section {  
  margin: 0 auto;  
}  
@media (min-width: 768px) {  
  .main-section {  
    padding: 6px 12px;  
    width: 900px;  
  }  
}
```

Wywołujemy **mixin tablet**.

Przekazujemy do niego tzw. **content**.



# Mobile First

# Mobile First

**Mobile first** to zalecane podejście, w którym definiowanie wyglądu rozpoczyna się od najprostszego do coraz bardziej zaawansowanego. Wraz ze wzrostem szerokości ekranu style powinny być dodawane dopiero, jeśli są potrzebne.

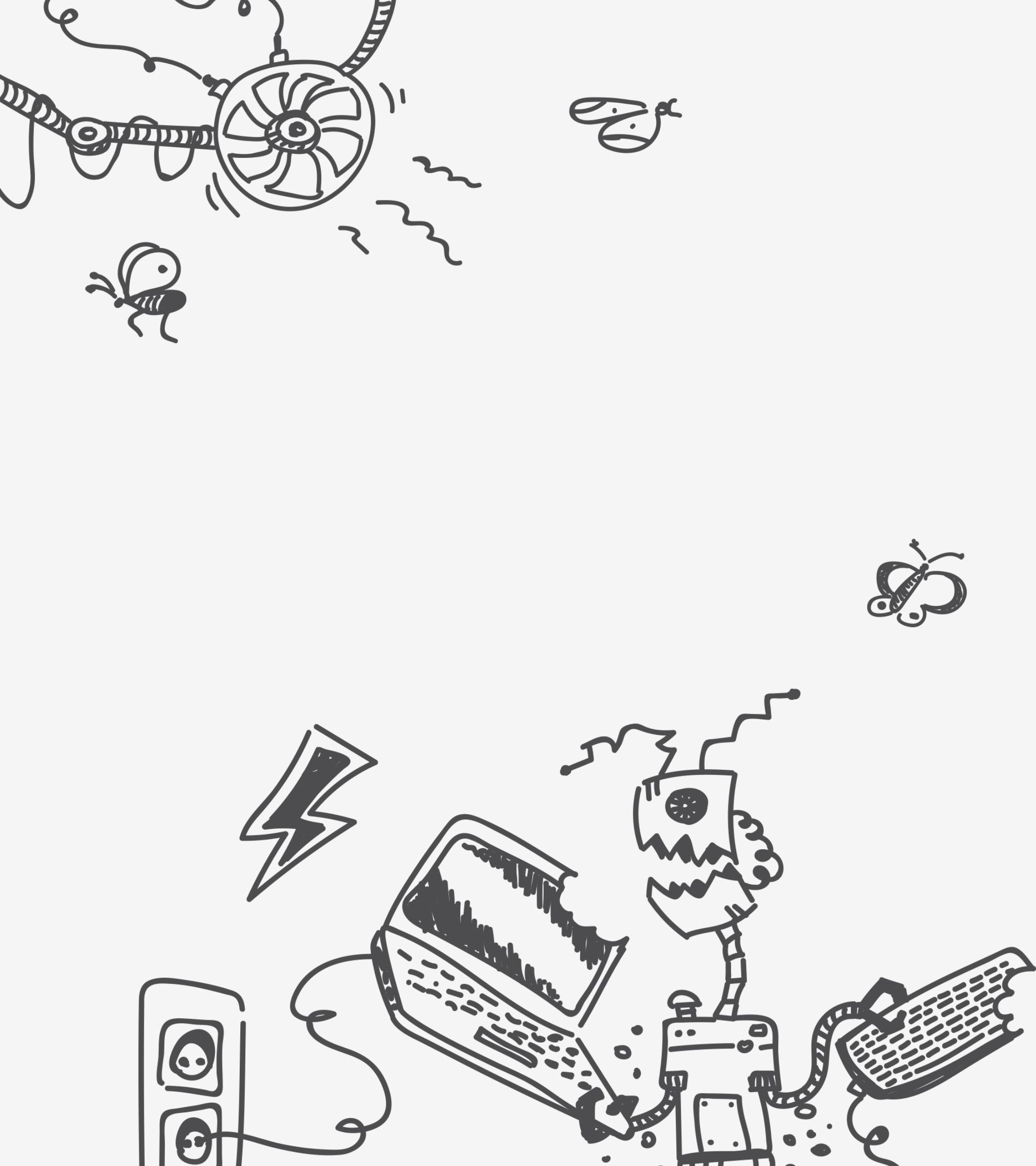
```
.column {  
    width: 100%;  
  
    @media only screen and (min-width: 600px) {  
        width: 50%;  
    }  
  
    @media only screen and (min-width: 1200px) {  
        width: 33.333%;  
    }  
}
```

# Podejście mobile first w Foundation Framework

```
/* Small screens */
@media only screen { } /* Define mobile styles */
@media only screen and (max-width: 40em) { }

/* Medium screens */
@media only screen and (min-width: 40.063em) { }
@media only screen and (min-width: 40.063em)
and (max-width: 64em) { }

/* Large screens */
@media only screen and (min-width: 64.063em) { }
@media only screen and (min-width: 64.063em)
and (max-width: 90em) { }
```



# RWD Flexbox

# Wprowadzenie

## Flexbox

Korzystając z flexboxa do układania elementów na stronie doświadczałyśmy sytuacji, że elementy na stronie działały bardzo elastycznie. Za pomocą takich właściwości jak flex-direction lub flex-wrap, mogliśmy manewrować układaniem elementów w pionie i poziomie, oraz ustawiać warunki, w których elementy mają spadać pod siebie.

Teraz mając wiedzę o media queries oraz o technikach RWD możemy nauczyć się jak połączyć naszą wiedzę w całość, aby tworzyć całościowe układy stron internetowych, dostosowane zarówno do małych telefonów jak i dużych ekranów komputera.

W tym rozdziale poznamy kilka strategii i zasad jak korzystać z technologii flexbox przy mniejszych urządzeniach.

# Kiedy używać flexbox?

## Jak używać techniki flexbox?

Flexbox przyda się w następujących sytuacjach:

- do ustawiania elementów obok siebie
- do ustawiania maksymalnych odstępów między elementami (np. wertykalnie lub horyzontalnie względem kontenera). Do tego potrzebujemy skorzystać z właściwości justify-content i align-items)
- do układania elementów pod sobą (flex-wrap)

Oznacza to, że flexbox będzie potrzebny raczej do kontenerów (np. całej nawigacji, czy całych boksów) niż do poszczególnych elementów.

Warto przyjąć zasadę, że używamy techniki flexbox wtedy kiedy nie możemy osiągnąć pożądanego efektu za pomocą elementów inline - blokowych (np. gdy potrzebujemy ustawić dwa elementy obok siebie, nadać im paddingi, lub gdy potrzebujemy aby elementy były w idealnym odstępie między sobą)

# Kiedy nie używać flexboxa?

## Jak nie używać flexboxa

Flexboxa nie należy stosować w poniższych sytuacjach:

- kiedy możemy osiągnąć ten sam efekt za pomocą właściwości - display block, lub inline-block
- kiedy element musi być sztywnie ustawiony względem jakiegoś elementu (np. ozdobniki do list). Wtedy lepiej użyć pozycji relatywnej/absolutnej

Warto pamiętać, że flexbox zmienia zachowanie dzieci. Zbyt częste używanie tej właściwości powoduje, że nasz kod przestaje być czytelny i zrozumiały.

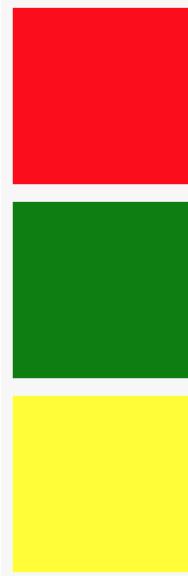
Przy tworzeniu CSS warto stosować zasadę, że im mniej kodu tym lepiej. Oznacza to, że powinniśmy używać tylko tych właściwości, które nam są niezbędne.

# Strategia - warunkowe renderowanie

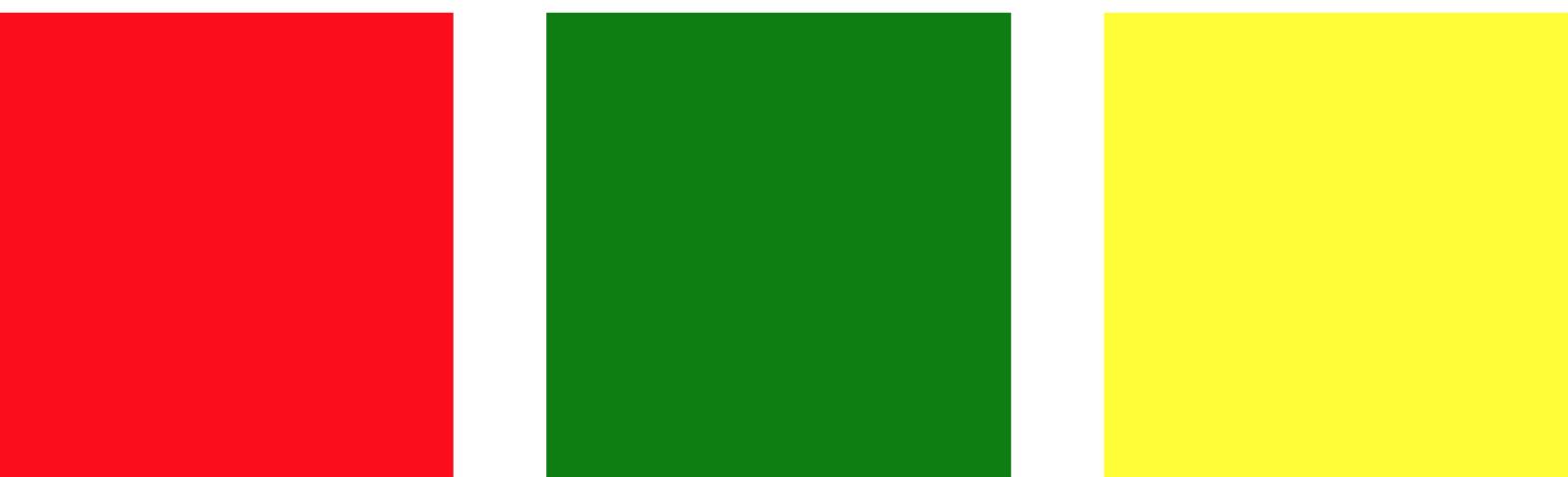
Jeśli elementy powinny znajdować się obok siebie tylko na dużym ekranie, a na mniejszych ekranach chcemy, aby wszystko było jedno pod drugim - wtedy warto ustawić `display: flex` tylko dla dużych ekranów.

```
.container {  
    display: block;  
    /* domyślna wartość */  
  
    @media screen and (min-width: 640px) {  
        display: flex;  
    }  
}
```

Mobile:



Desktop:



# Strategia - spadanie elementów

Pierwszą strategią RWD przy budowaniu układów stron jest wyśrodkowanie zawartości kontenera dla **ekranów mobilnych** za pomocą **justify-content**. Na dużych ekranach zazwyczaj możemy pozwolić sobie na układanie elementów obok siebie, natomiast na telefonach komórkowych musimy elementy ułożyć jeden pod drugim.

Na następnym slajdzie znajdują się zdjęcia opisujące tę sytuację.

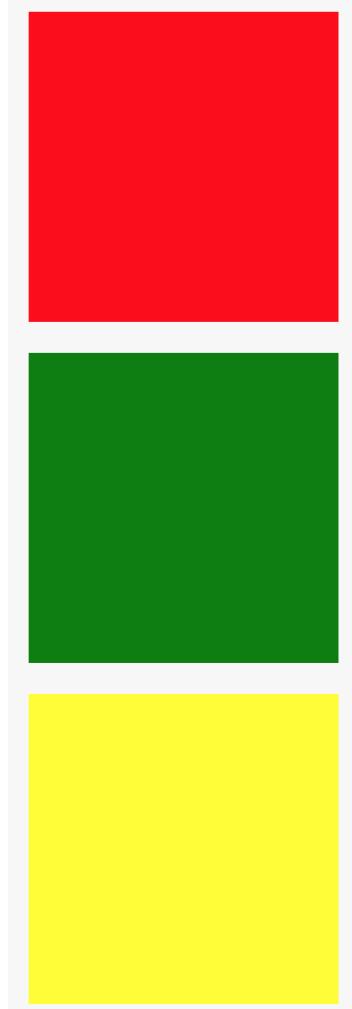
```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    justify-content: center;  
  
    @media screen and (min-width: 640px) {  
        justify-content: flex-start;  
    }  
}
```

# Strategia - spadanie elementów c.d.

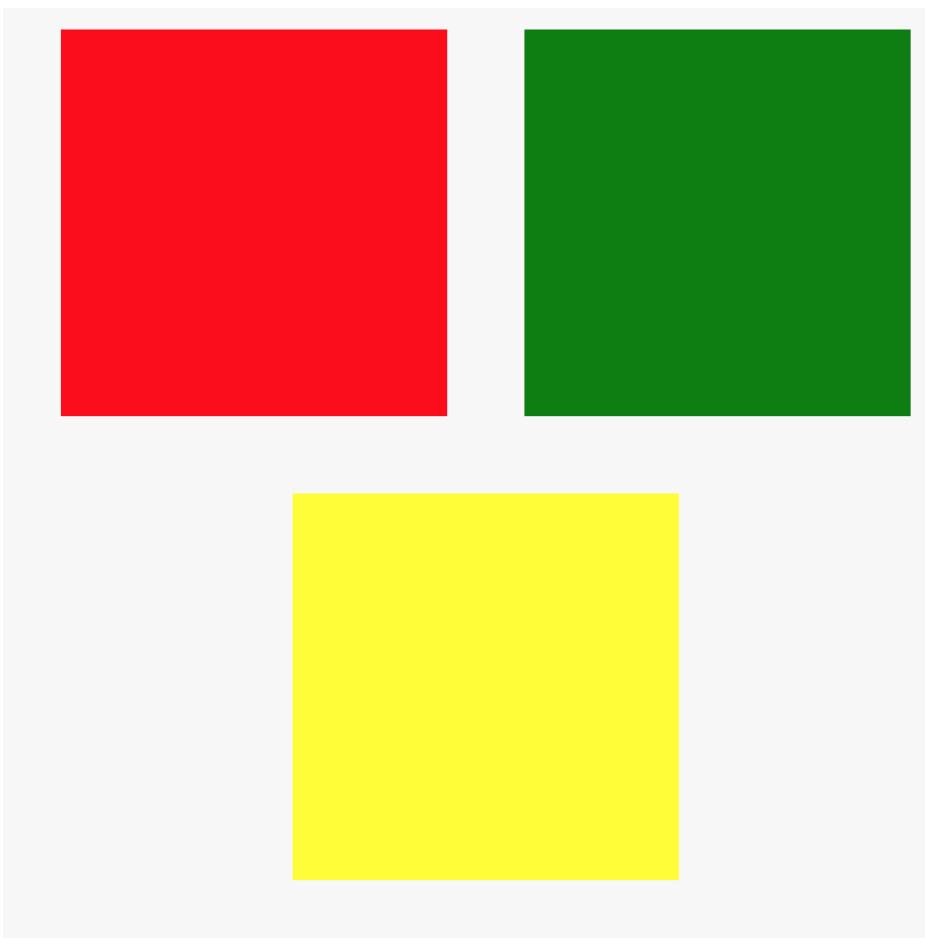
Desktop:



Mobile:



Tablet:



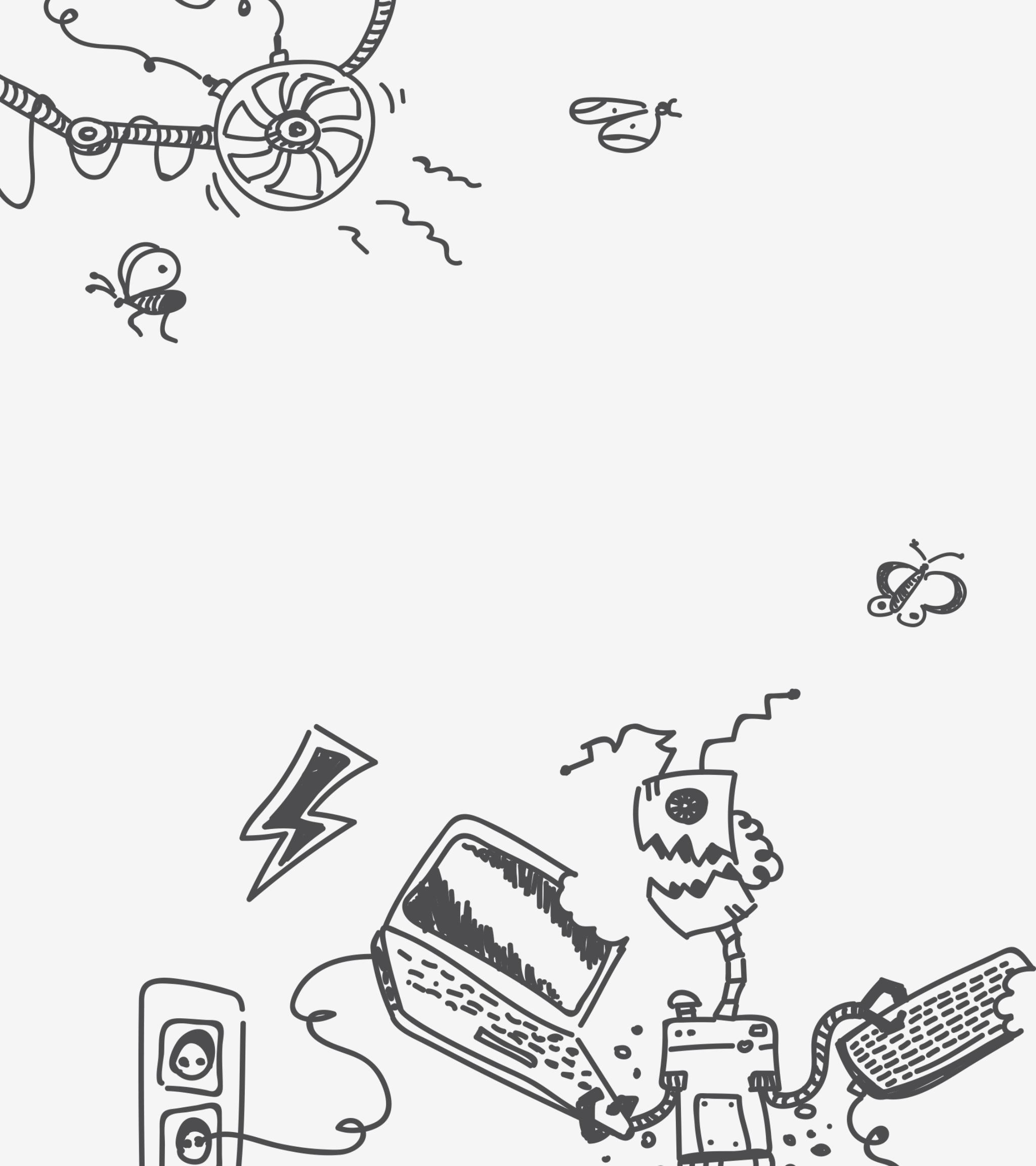
# Podsumowanie

## Jak ćwiczyć układanie stron?

Nic nie jest w stanie zastąpić praktyki przy tworzeniu układów stron internetowych.

Tutaj znajduje się zbiór kilkudziesięciu układów stron internetowych, które w ramach treningu możemy zakodować przy pomocy HTML/CSS.

<https://freshdesignweb.com/free-psd-website-templates/>



Na czym polega  
RWD?

# Na czym polega responsywność?

## Responsive Web Design

- RWD to technika projektowania stron WWW tak, aby ich wygląd i układ dostosowywał się automatycznie do rozmiaru okna urządzenia, na którym jest wyświetlany.
- Strona tworzona w tej technice jest uniwersalna i wyświetla się dobrze zarówno na dużych ekranach, jak i smartfonach czy tabletach.
- Użytkownik – dzięki RWD – może optymalnie korzystać ze strony.

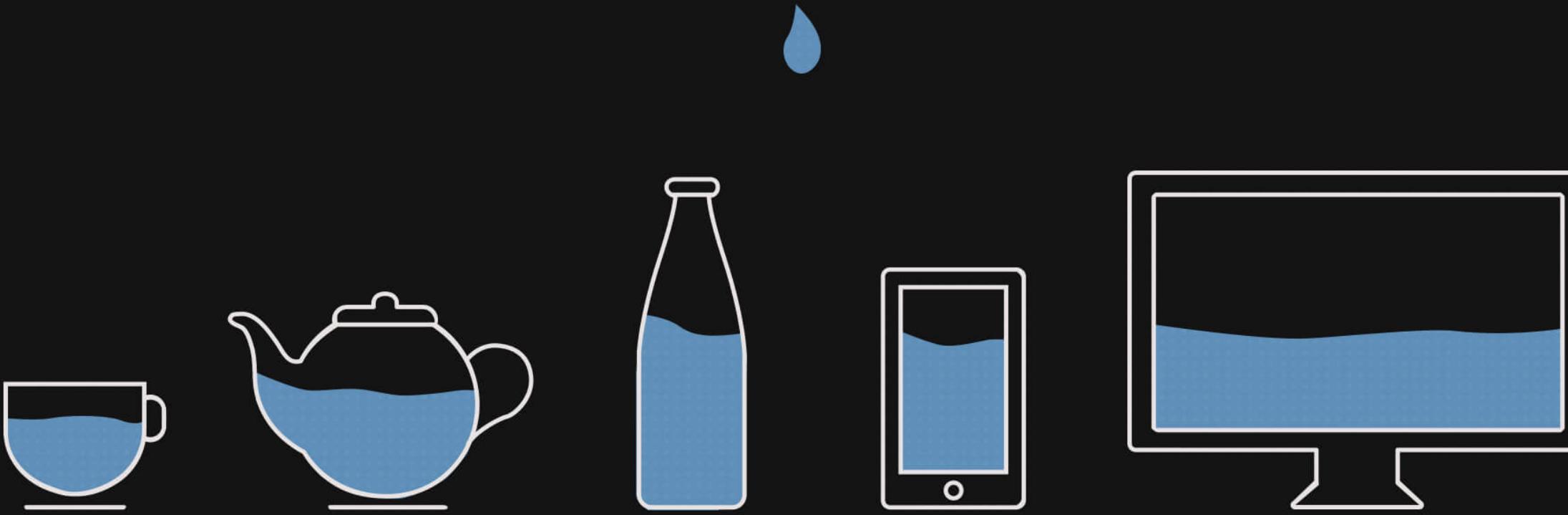
# Na czym polega responsywność?

## Responsive Web Design

- W ogólnym założeniu ten sam kod HTML powinien być serwowany dla wszystkich urządzeń a dopasowanie powinno zostać osiągnięte przez zmiany w CSS.
- Responsywność przede wszystkim, polega na **przystosowaniu poszczególnych elementów / contentu** serwisu internetowego **do konkretnego urządzenia** z różnymi rozdzielczościami / właściwościami ekranu.

- Po poprawnym wykonaniu ma ułatwić użytkownikowi korzystanie z serwisu / aplikacji.

# CONTENT IS LIKE WATER



“ You put water into a cup it becomes the cup.  
You put water into a bottle it becomes the bottle.  
You put it in a teapot, it becomes the teapot. ”

---

Josh Clark (*originally Bruce Lee*) - Seven deadly mobile myths

Illustration by Stéphanie Walter

# RWD a Adaptive Web Design

## RWD

- Do ustalania szerokości elementów używa się `%`, `rem`.
- Kontener opakowujący - nadający szerokość strony - nie jest blokowany właściwością `width` np. (960px) a jedynie `max-width`, który ogranicza szerokość powyżej jakiejś wartości.
- Największym plusem jest swobodna możliwość rozplanowania elementów na stronie.
- Można manipulować zawartością strony, bez białych, pustych miejsc ekranu po obu stronach.

## AWD

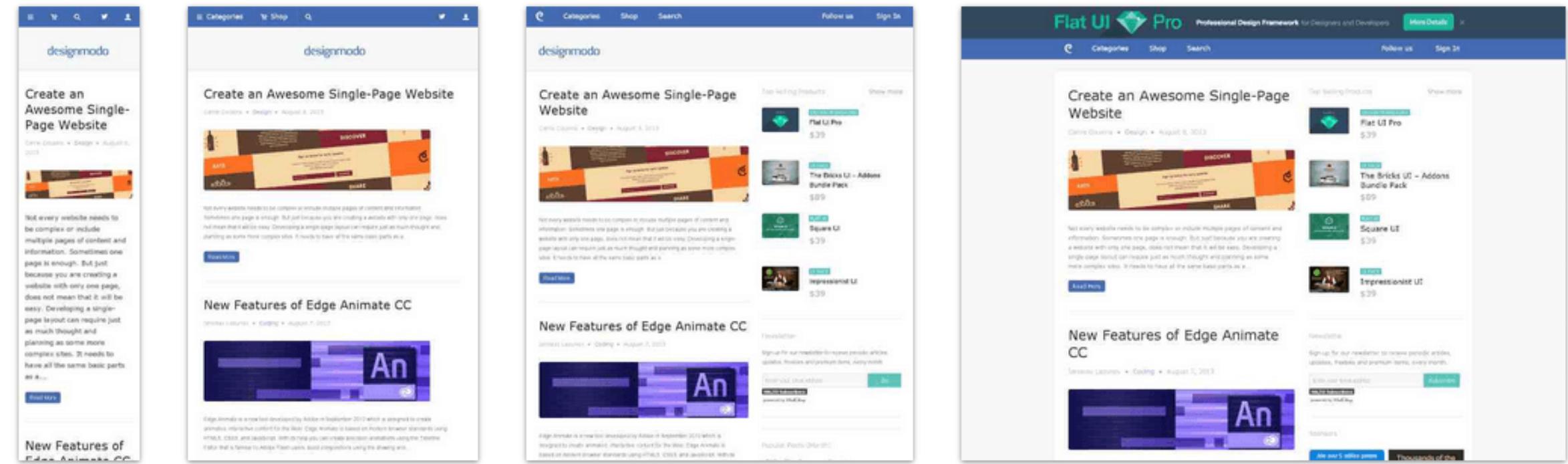
- Kontener opakowujący posiada właściwość `width` (określoną w px) dla różnych szerokości zdefiniowanych w `media query`.
- Dużym plusem jest idealne rozplanowanie elementów na projekcie graficznym dla poszczególnych szerokości ekranu
- Mamy ograniczoną możliwość rozplanowania elementów na stronie.

# Responsywność

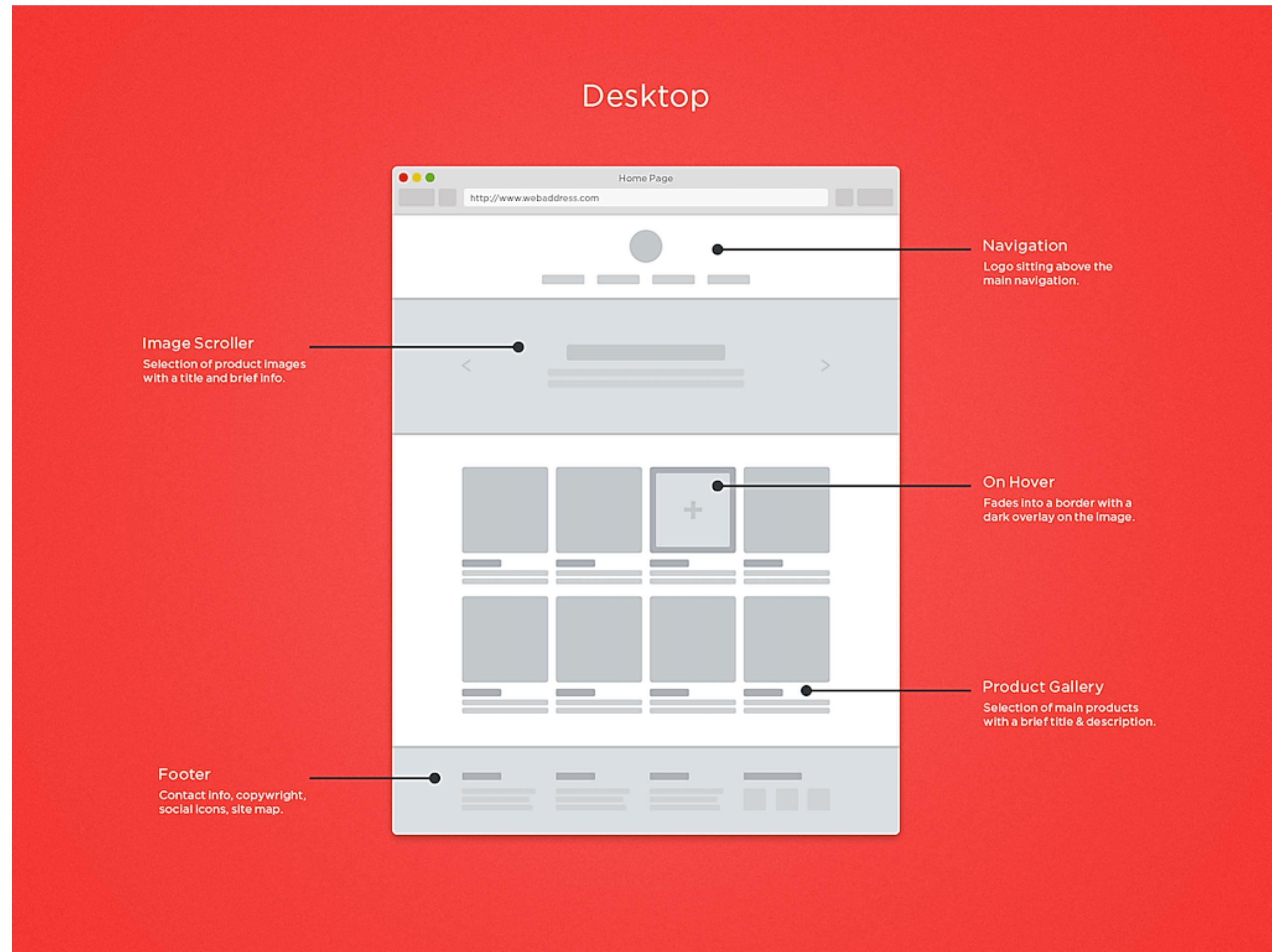
The screenshots illustrate the Smashing Magazine website's responsive design across different devices. The left screenshot shows the desktop homepage with a sidebar containing categories like CODING, DESIGN, MOBILE, GRAPHICS, UX DESIGN, and WORDPRESS. The middle screenshot shows a single article page for "This Interface Is A (Good) Joke!" with a sidebar menu. The right screenshot shows a newsletter sign-up page with a sidebar containing book recommendations.

<https://www.smashingmagazine.com/>

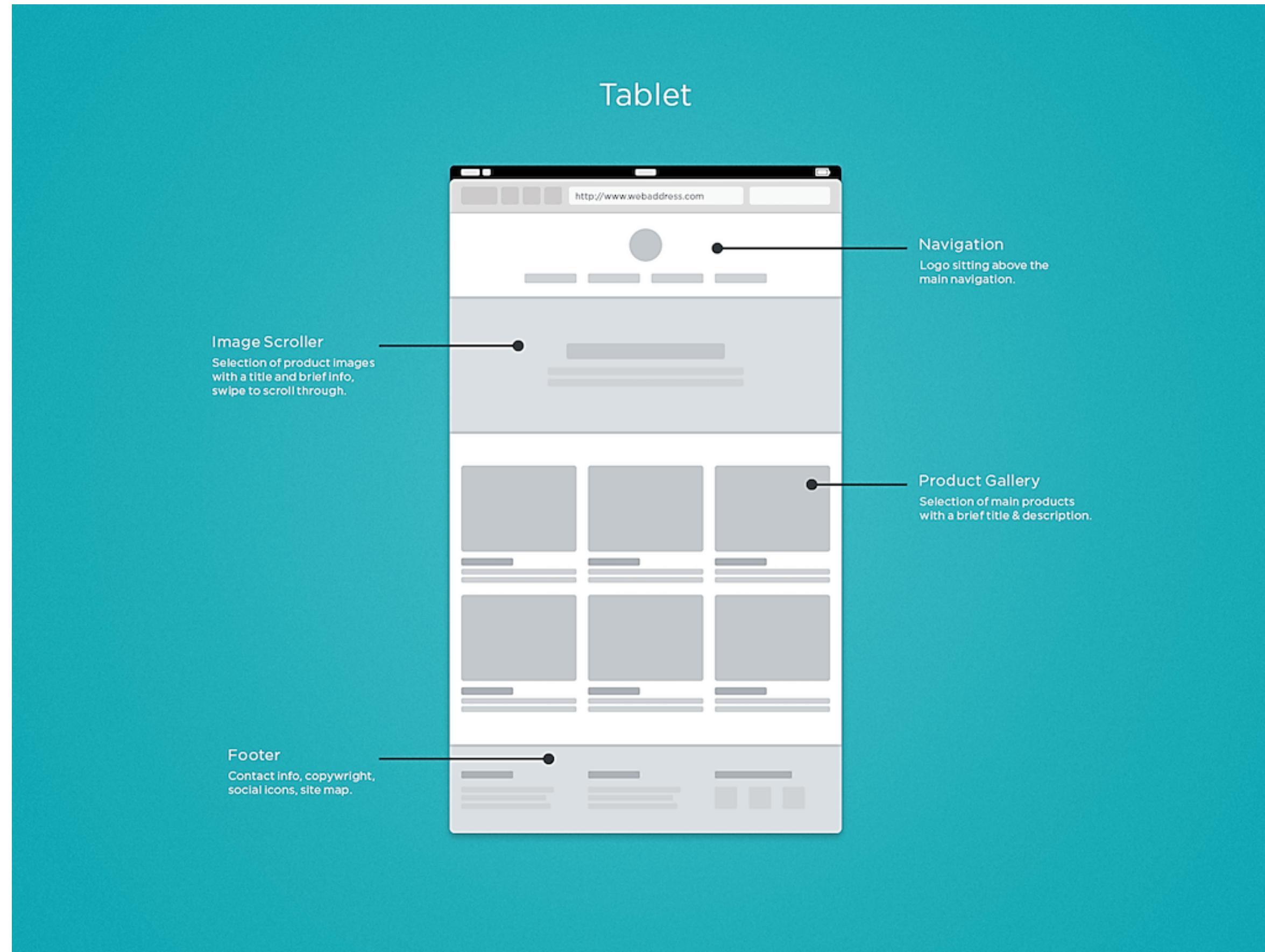
# Responsywność a szerokość ekranu



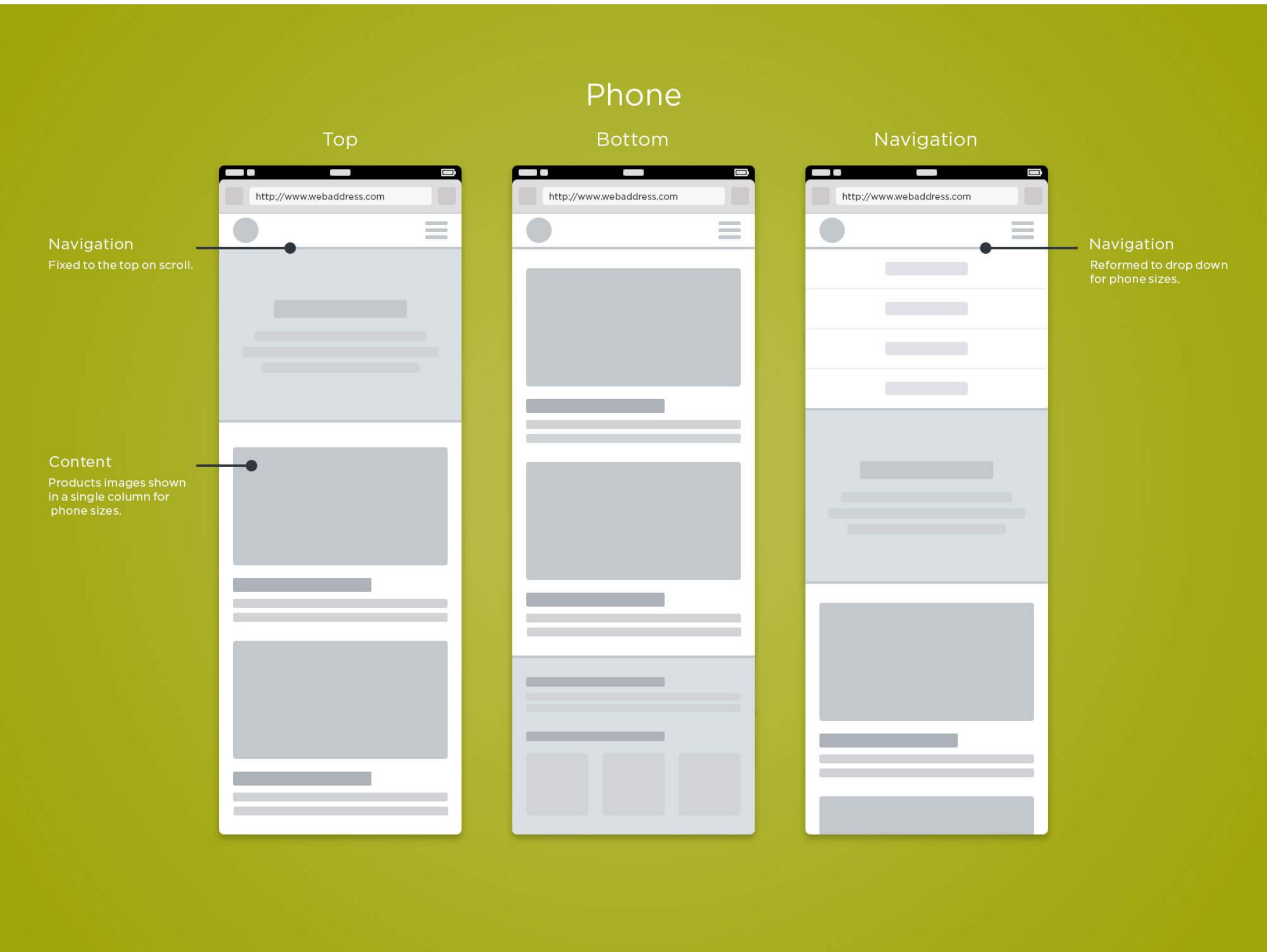
# Desktop



# Tablet



# Mobile



# RWD – założenia i istota

## Założenia

- CSS3 Media Queries.
- Elastyczny układ strony z reguły oparty na gridzie i jednostkach relatywnych (`em`, `rem`, `%`).

- Elastyczne media dostosowujące swój rozmiar do możliwości ekranu, odpowiednio zoptymalizowane.

# RWD – założenia i istota

## Dlaczego jest istotna?

- Urządzenia, dzięki którym możemy korzystać z Internetu, opierają się na bardzo różnej interakcji z użytkownikiem. Wyświetlanie na nich treści w ten sam sposób często jest utrudnione lub niemożliwe.
- Ponad połowa ruchu generowanego w Internecie pochodzi z urządzeń mobilnych.
- Właściciel witryny powinien zadbać o to, aby strona była dostosowana do urządzeń mobilnych.
- Google daje wyższy ranking stronom responsywnym niż serwisom bez tej technologii (słowo klucz - **mobilegeddon**).

# Responsywność - wady i zalety

## Jakie ma zalety?

- Brak problematycznych przekierowań wydłużających czas ładowania strony.
- Łatwiejsze udostępnianie tego samego adresu.
- Mniej czasochłonne zarządzanie stroną.
- Oszczędność zasobów robotów skanujących strony, łatwiejsza indeksacja jednej strony zamiast kilku jej wersji.

## Jakie ma wady?

- Możliwy wzrost transferu danych.
- Problemy z kompatybilnością różnych przeglądarek.
- Trudności w dobraniu odpowiednich przedziałów szerokości ekranów.

# RWD i progressive enhancement

## Główne idee

- Niektórzy rozszerzają trzy główne założenia RWD o **progressive enhancement (PE)**.
- Ideą **progressive enhancement** jest to, aby treść strony była dostępna na wszystkich urządzeniach a wygląd i interakcje rozszerzały się dopiero wtedy, jeśli urządzenie jest w stanie je obsłużyć.
- Wszystko powinno odbywać się bez obciążania urządzeń o mniejszych możliwościach.

## Niezależność warstw

- Każda z warstw powinna być całkowicie niezależna od innej.
- Treść + semantyka nie powinny wiedzieć nic o prezentacji.
- Prezentacja nie powinna wiedzieć nic o warstwie zachowania.

Lepiej coś użytkownikowi pokazać, niż odesłać go z niczym.

# progressive enhancement

## HTML

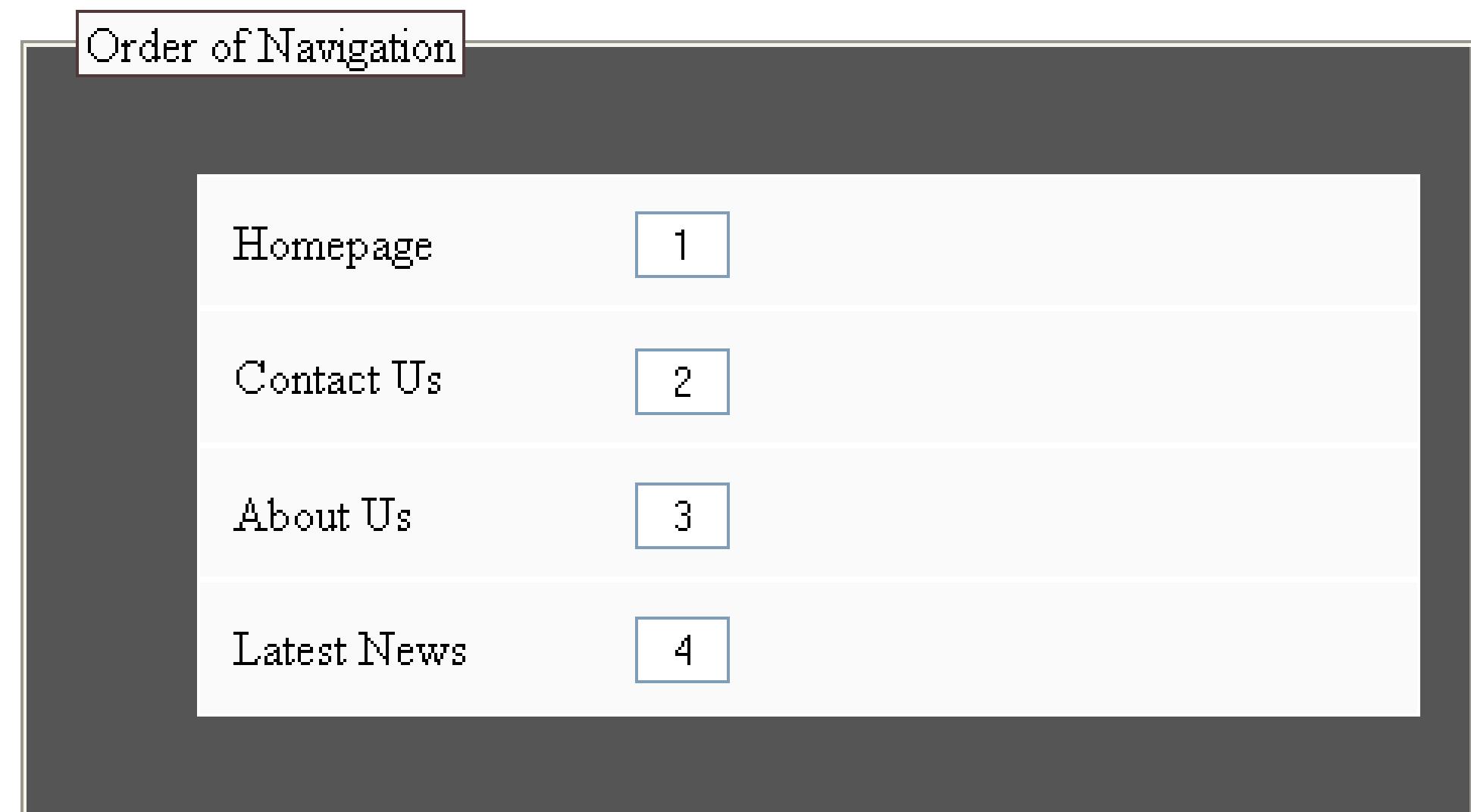
Order of Navigation

1. Homepage	Change the order for Homepage	1
2. Contact Us	Change the order for Contact Us	2
3. About Us	Change the order for About Us	3
4. Latest News	Change the order for Latest News	4

Save new order

# progressive enhancement

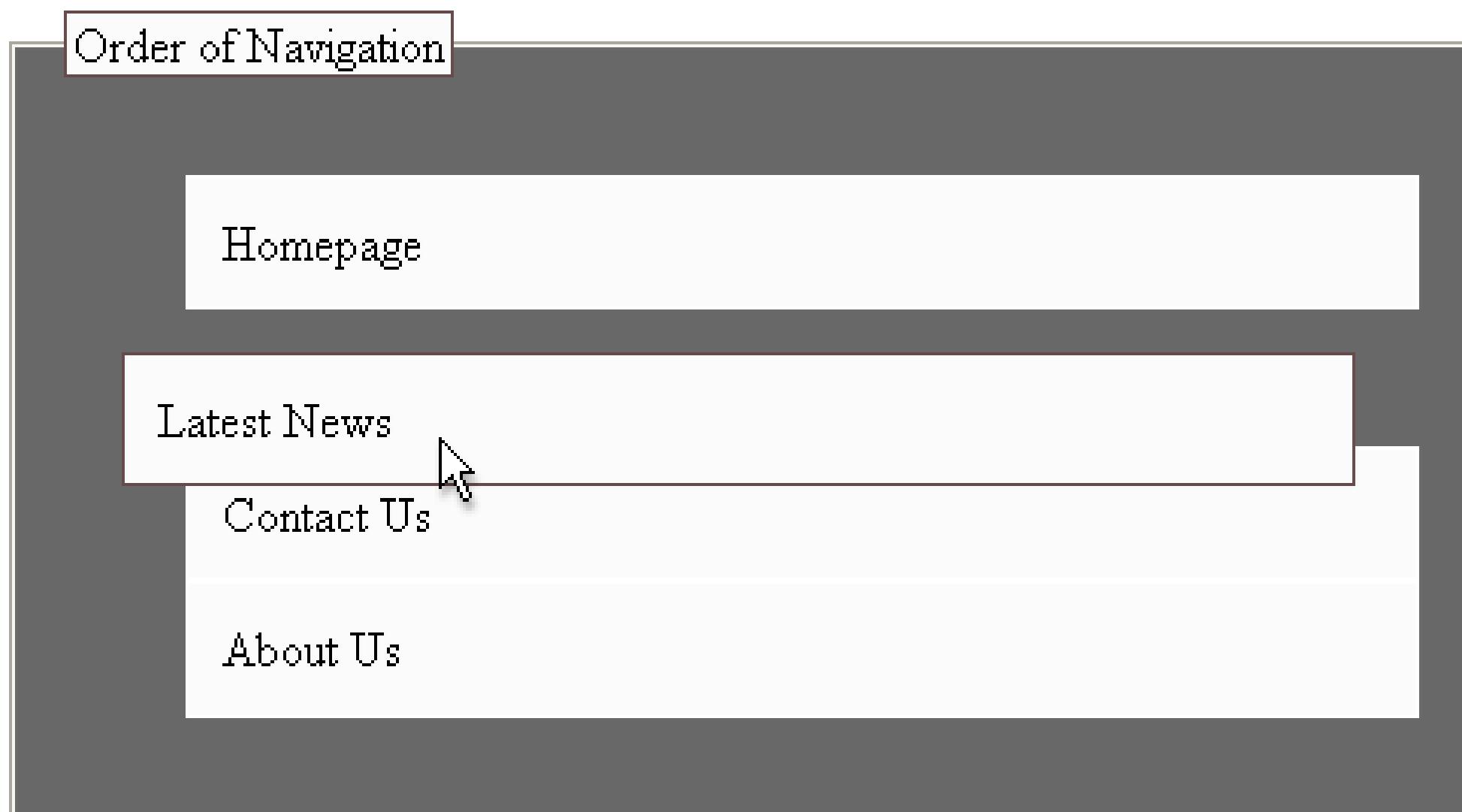
CSS



Save new order

# progressive enhancement

## JavaScript



# Osobna wersja mobilna

Innym podejściem jest przygotowanie **osobnej wersji strony** dla różnych urządzeń pod innym adresem URL, np. [m.facebook.com](http://m.facebook.com). Niesie to za sobą koszt wdrożenia i utrzymania nawet kilku różnych wersji (np. na tablet, Smart TV itp).

Osobne wersje umożliwiają lepszą optymalizację wczytywanych zasobów i odpowiednie dobranie dostępnych dla użytkownika funkcji.

Przy bardzo rozbudowanych serwisach takie rozwiązanie jest optymalne.

# Front-end a aplikacje mobilne

Można też stworzyć aplikację aby użytkownik posiadał dodatkowy kanał, przez który może konsumować treści. Teoretycznie takie rozwiązanie wymaga od twórcy znajomości jednej z technologii mobilnych np. Java, Objective-c, itp.

Jednak od kilku lat mamy możliwość tworzenia tzw. aplikacji hybrydowych. Za pomocą odpowiedniego frameworka konwertujemy projekt stworzony np. w HTML, CSS i JS. Po konwersji projekt działa na iOS, Android oraz Windows Phone. Nie jest to idealne rozwiązanie m.in. ze względu na szybkość działania aplikacji.

Najbardziej popularne frameworki do budowania aplikacji hybrydowych:

- <https://cordova.apache.org/>
- <http://ionicframework.com/>
- <https://reactnative.dev/>
- <https://nativescript.org/>
- <https://flutter.io/>

# Podsumowanie rozdziału

## O czym należy pamiętać?

- RWD to technika projektowania stron WWW tak, aby ich wygląd i układ dostosowywał się automatycznie do rozmiaru okna urządzenia, na którym jest wyświetlany.
- Lepiej użytkownikowi coś pokazać niż odesłać go z niczym (Progressive Enhancement)
- RWD, czy osobna wersja mobilna? To zależy od skali projektu.

- Możemy wersję mobilną aplikacji stworzyć za pomocą HTML, CSS i JavaScript – jest to wtedy aplikacja hybrydowa.
- Używamy jednostek relatywnych np. %, em lub rem.



# Jednostki w RWD

# Em a font-size

## Em

Zacznijmy od tego, że użycie `em` w `font-size`, odnosi się do najbliższego rodzica a nie do tego samego elementu. Czyli wielkość czcionki jest obliczana na podstawie czcionki rodzica.

Zobacz przykład:

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
}
```

```
.child {  
    font-size: 2em;  
}
```

# Em a font-size

## Em

Zacznijmy od tego, że użycie `em` w `font-size`, odnosi się do najbliższego rodzica a nie do tego samego elementu. Czyli wielkość czcionki jest obliczana na podstawie czcionki rodzica.

Zobacz przykład:

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
}
```

Jestem rodzicem - 10px.

```
.child {  
    font-size: 2em;  
}
```

# Em a font-size

## Em

Zacznijmy od tego, że użycie `em` w `font-size`, odnosi się do najbliższego rodzica a nie do tego samego elementu. Czyli wielkość czcionki jest obliczana na podstawie czcionki rodzica.

Zobacz przykład:

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
}
```

Jestem rodzicem - 10px.

```
.child {  
    font-size: 2em;  
}
```

Jestem dzieckiem - **20px** (bo 10px x 2).

# Em a inne wielkości

## Em

Ale definicja em mówi, że jest to jednostka relatywna odnosząca się do wielkości czcionki tego elementu, na którym jest użyta!

Zobacz przykład:

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
}
```

```
.child {  
    font-size: 2em;  
    width: 2em;  
    height: 4em;  
    border: 1px solid red;  
}
```

# Em a inne wielkości

## Em

Ale definicja em mówi, że jest to jednostka relatywna odnosząca się do wielkości czcionki tego elementu, na którym jest użyta!

Zobacz przykład:

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

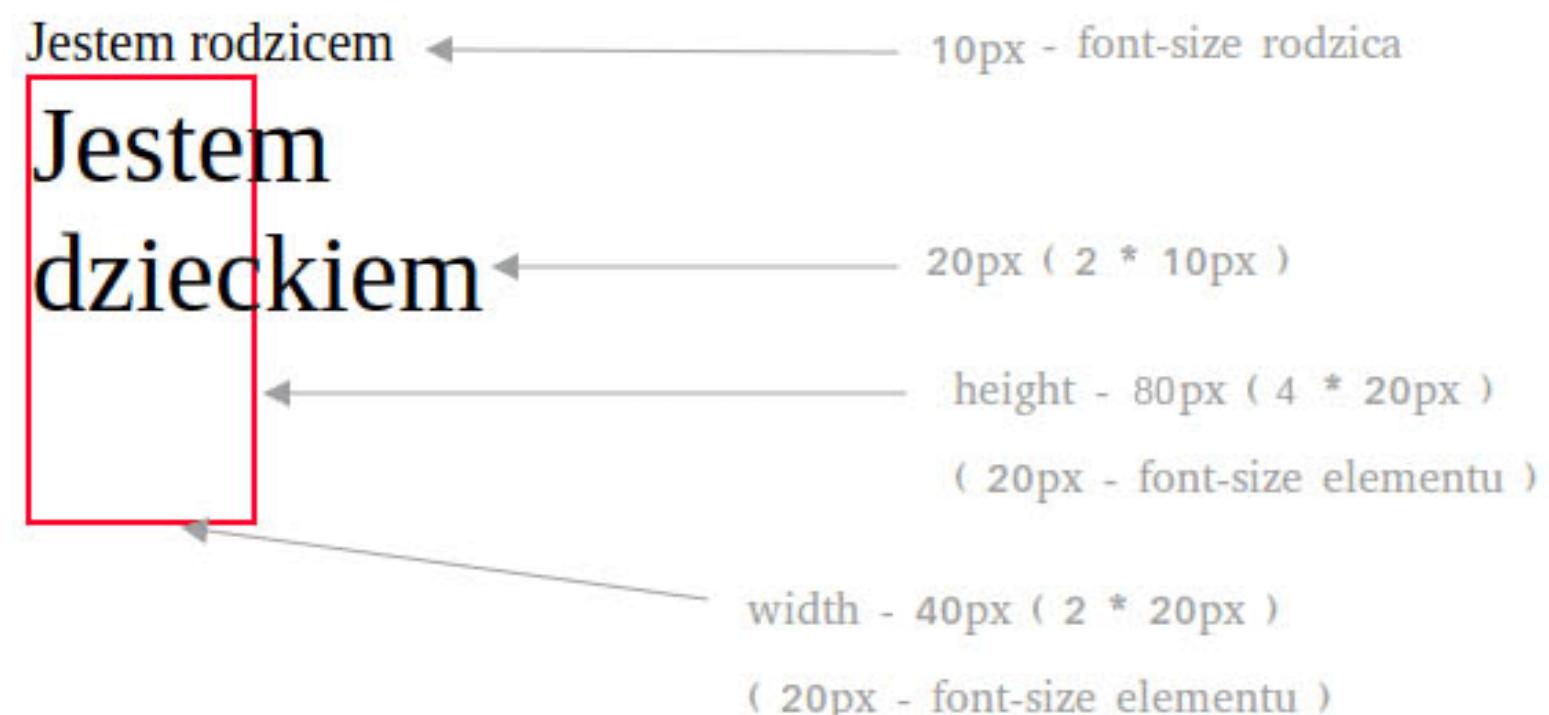
## CSS

```
.parent {  
    font-size: 10px;  
}
```

Jestem rodzicem - 10px.

```
.child {  
    font-size: 2em;  
    width: 2em;  
    height: 4em;  
    border: 1px solid red;  
}
```

# Em - podsumowanie



## Podsumowując em

Jednostka ta odnosi się do wielkości czcionki danego elementu czyli wszystkie wielkości ustawione w em są obliczane na podstawie własności `font-size` tego elementu.

**Wyjątkiem** jest samo ustawianie `font-size`. W takim przypadku wielkość jest obliczana na podstawie rodzica.

# Rem a font-size

## Rem

Jednostka `rem` jest również jednostką relatywną. Jest natomiast obliczana na podstawie wielkości czcionki użytej w elemencie `root`, którym zazwyczaj jest element `html`.

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## SCSS

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 2rem;  
    width: 2rem;  
    border: 1px solid red;  
}
```

# Rem a font-size

## Rem

Jednostka `rem` jest również jednostką relatywną. Jest natomiast obliczana na podstawie wielkości czcionki użytej w elemencie `root`, którym zazwyczaj jest element `html`.

## HTML

```
<div class="parent">
    Jestem rodzicem
    <div class="child">
        Jestem dzieckiem
    </div>
</div>
```

## SCSS

```
html {
    font-size: 20px;
}

.parent {
    font-size: 10px;
}

.child {
    font-size: 2rem;
    width: 2rem;
    border: 1px solid red;
}
```

Ten element będzie rootem.

# Rem a font-size

## Rem

Jednostka `rem` jest również jednostką relatywną. Jest natomiast obliczana na podstawie wielkości czcionki użytej w elemencie `root`, którym zazwyczaj jest element `html`.

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## SCSS

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 2rem;  
    width: 2rem;  
    border: 1px solid red;  
}
```

Ten element będzie rootem.  
Jestem rodzicem - `10px`.

# Rem a font-size

## Rem

Jednostka `rem` jest również jednostką relatywną. Jest natomiast obliczana na podstawie wielkości czcionki użytej w elemencie `root`, którym zazwyczaj jest element `html`.

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## SCSS

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 2rem;  
    width: 2rem;  
    border: 1px solid red;  
}
```

5

Ten element będzie rootem.

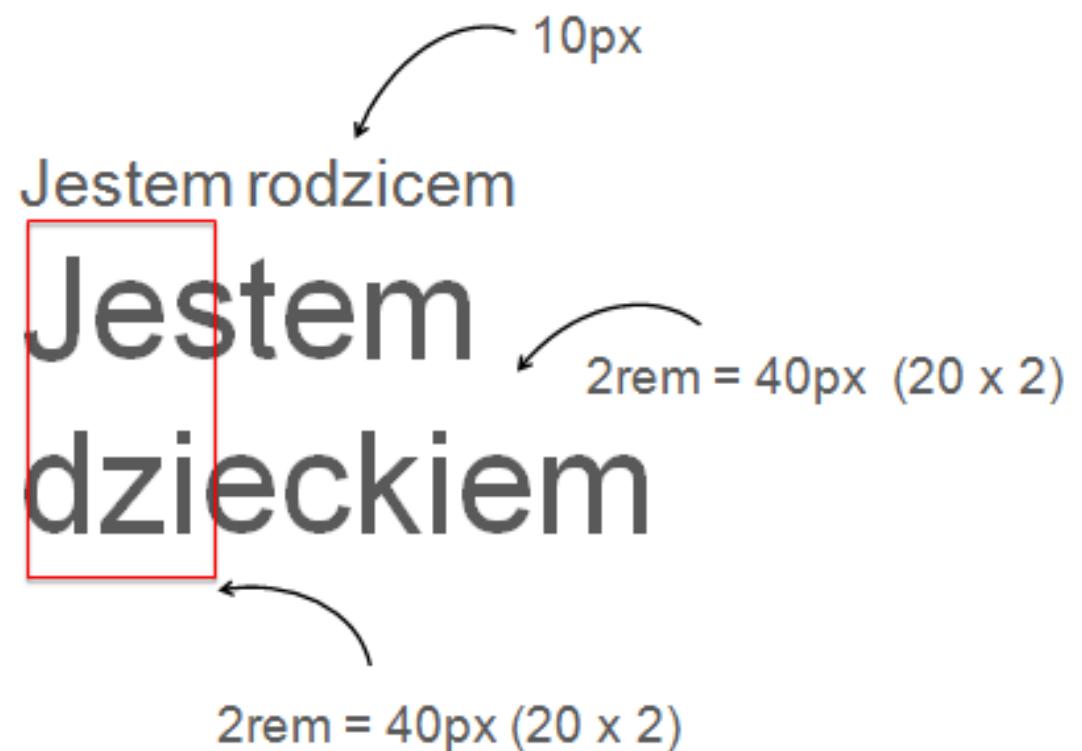
Jestem rodzicem - `10px`.

Ten element będzie dzieckiem - `40px`  
(bo  $20 \times 2$ ).

# Rem a inne wielkości

## Rem

Podobnie jeśli chodzi o ustawianie innych wielkości w rem, również wartości są obliczane na podstawie wielkości czcionki ale z elementu root.



## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

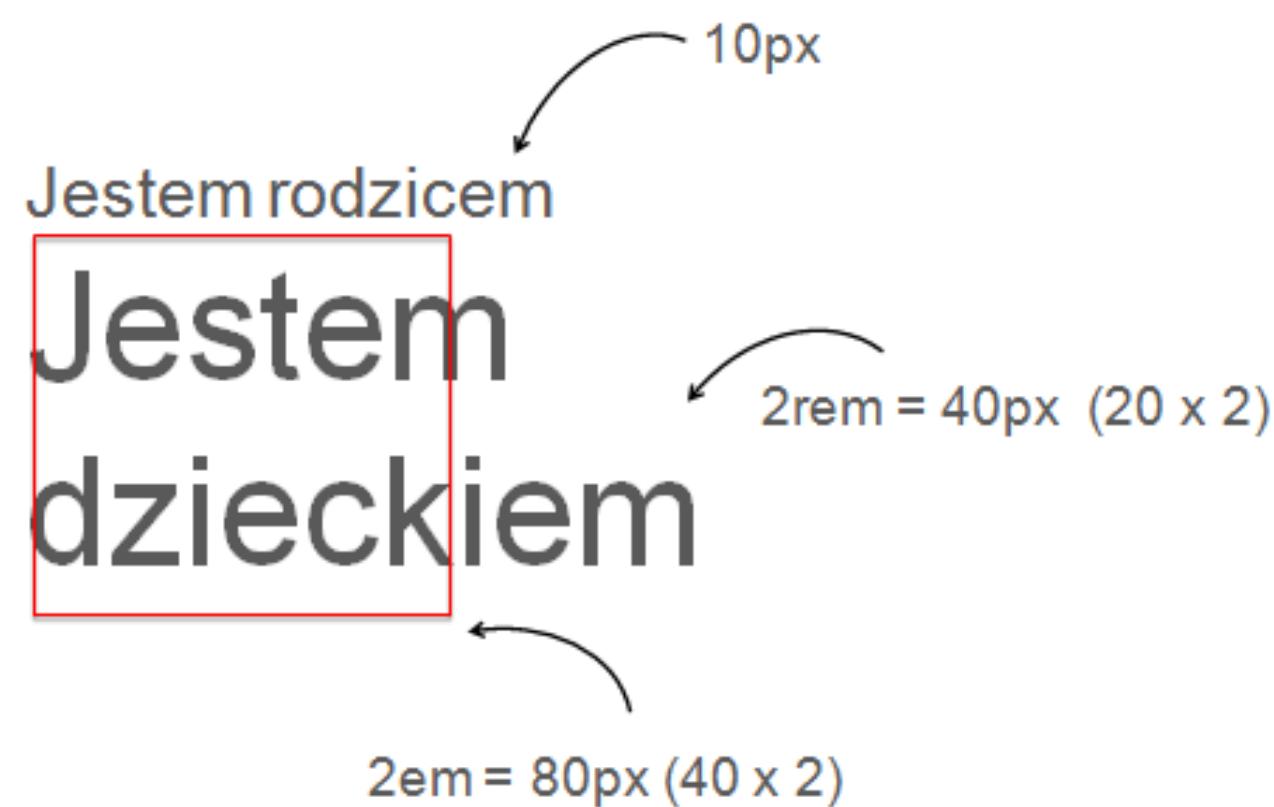
## CSS

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 2rem;  
    width: 2rem;  
    border: 1px solid red;  
}
```

# Em razem z Rem

## Em użyte razem z rem

Tutaj zasada się nie zmienia. Em w przypadku innych właściwości niż font-size będzie obliczany na podstawie wielkości czcionki danego elementu, nie ważne czy jest ona ustaliona za pomocą px czy rem.



## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 2rem;  
    width: 2em;  
    border: 1px solid red;  
}
```

# Procent – font-size

%

Jednostka % ustawia wielkość na podstawie najbliższego przodka - jeśli element **parent** ma ustawioną czcionkę na **10px** to element **child** z ustawioną czcionką na **200%** będzie ją miał 2 razy większą czyli **20px**.

**HTML**

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

**CSS**

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 200%;  
    width: 2rem;  
    border: 1px solid red;  
}
```

# Procent - font-size

%

Jednostka % ustawia wielkość na podstawie najbliższego przodka - jeśli element **parent** ma ustawioną czcionkę na **10px** to element **child** z ustawioną czcionką na **200%** będzie ją miał 2 razy większą czyli **20px**.

**HTML**

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

**CSS**

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 200%;  
    width: 2rem;  
    border: 1px solid red;  
}
```

Jestem rodzicem - **10px**.

# Procent – font-size

%

Jednostka % ustawia wielkość na podstawie najbliższego przodka - jeśli element **parent** ma ustawioną czcionkę na **10px** to element **child** z ustawioną czcionką na **200%** będzie ją miał 2 razy większą czyli **20px**.

**HTML**

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

**CSS**

```
html {  
    font-size: 20px;  
}  
.parent {  
    font-size: 10px;  
}  
.child {  
    font-size: 200%;  
    width: 2rem;  
    border: 1px solid red;  
}
```

Jestem rodzicem - **10px**.

Jestem dzieckiem - **20px** (bo  $10 \times 2$ ).

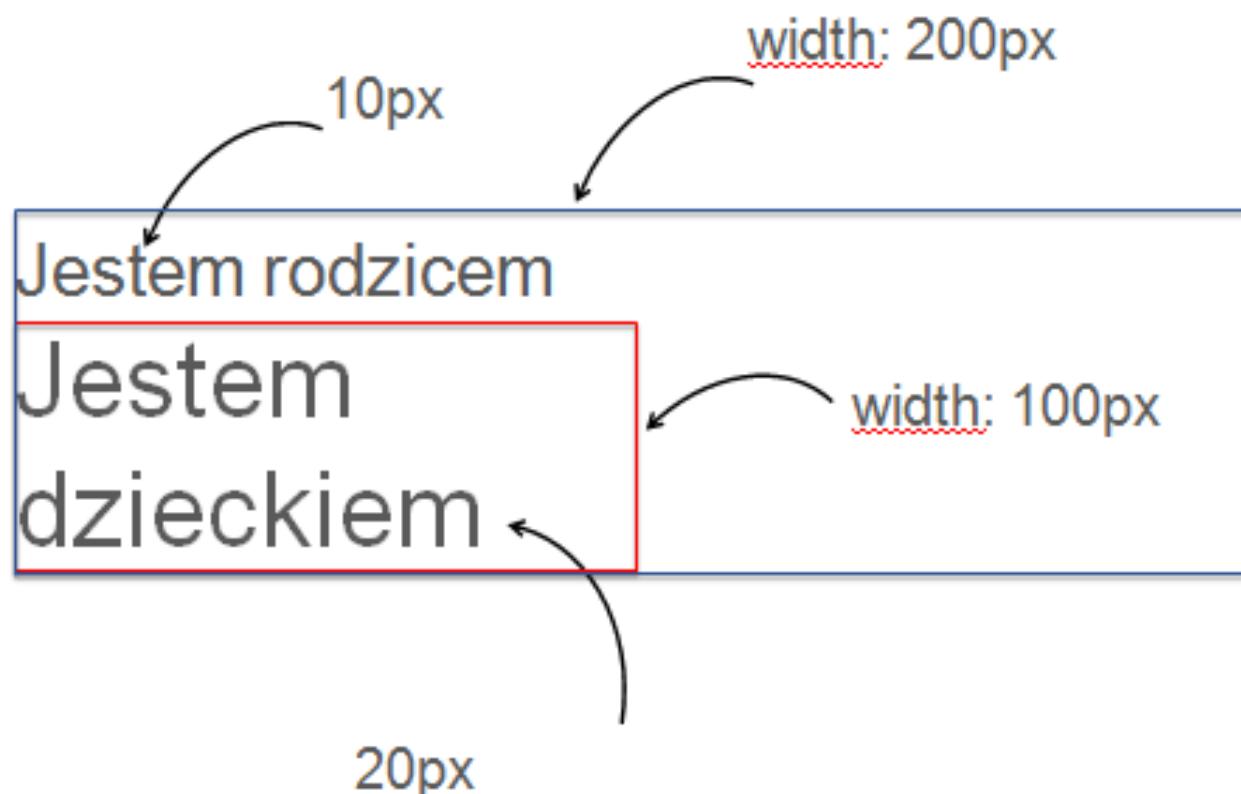
# Procent – width

## Określenie width w procentach

W przypadku width również wielkość jest obliczana na podstawie rodzica. Na przykładzie obok widać, że element

parent

ma ustawioną szerokość na 200px, a dziecko na 50%, co po obliczeniu daje 100px.



## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
    width: 200px;  
    border: 1px solid blue;  
}  
.child {  
    font-size: 200%;  
    width: 50%;  
    border: 1px solid red;  
}
```

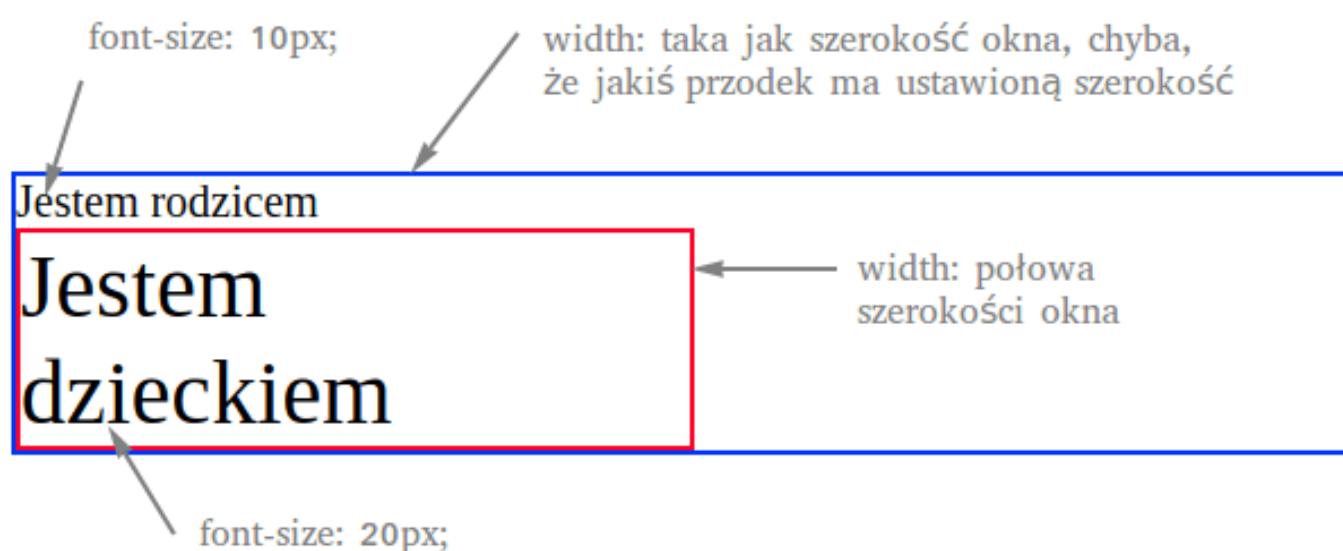
# Procent – width

## Określenie width w procentach

Jeśli nie ustawisz elementowi

**parent**

żadnej szerokości to będzie on zajmował całą dostępną szerokość rodzica (w naszym przypadku okno przeglądarki). Więc kiedy skorzystamy z jednostki procentowej w elemencie **child** obliczy on szerokość na podstawie szerokości okna. Chyba, że jeszcze jakiś przodek ma ustawioną szerokość.



## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

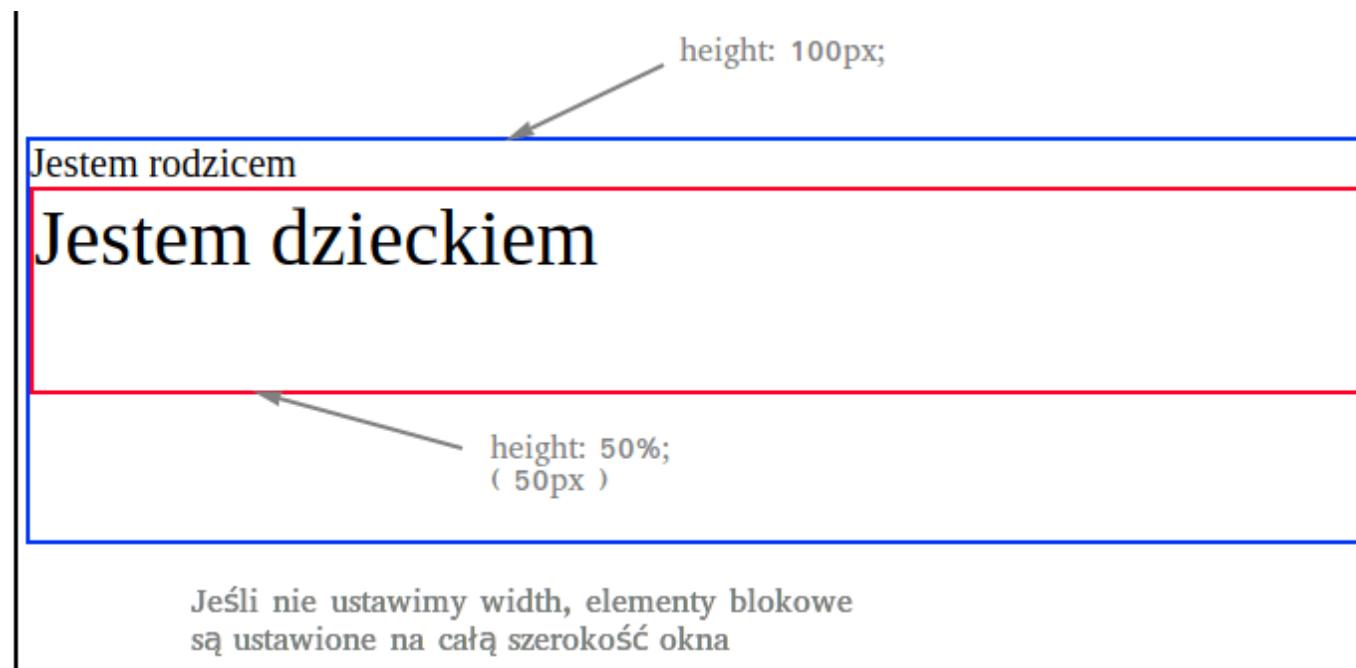
## CSS

```
.parent {  
    font-size: 10px;  
    border: 1px solid blue;  
}  
.child {  
    font-size: 200%;  
    width: 50%;  
    border: 1px solid red;  
}
```

# Procent – height

## Określenie height w procentach

Jeśli chodzi o własność `height` to tutaj bardzo ważne jest, aby najbliższy rodzic miał ustawioną konkretną wysokość.



## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
    height: 100px;  
    border: 1px solid blue;  
}  
.child {  
    font-size: 200%;  
    height: 50%;  
    border: 1px solid red;  
}
```

# Procent – height

## Określenie height w procentach

W przypadku jeśli nie ustawimy dla elementu

**parent**

wysokości, element

**child**

nie będzie wiedział o jaki procent nam chodzi.



## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
.parent {  
    font-size: 10px;  
    border: 1px solid blue;  
}  
.child {  
    font-size: 200%;  
    height: 50%;  
    border: 1px solid red;  
}
```

# Procent - height - ważne

## Określenie height w procentach

Ważne, żeby sobie zdać sprawę z tego, że w przypadku ustawiania

height

, brany jest pod uwagę **najbliższy rodzic** - jeśli ustawisz

height

dla przodka, który nie jest najbliższym rodzicem, wysokość nie będzie ustalona.

Jestem rodzicem

Jestem dzieckiem

height: ?????

## HTML

```
<div class="parent">  
    Jestem rodzicem  
    <div class="child">  
        Jestem dzieckiem  
    </div>  
</div>
```

## CSS

```
html, body {  
    height: 100%;  
}  
.parent {  
    font-size: 10px;  
    border: 1px solid blue;  
}  
.child {  
    font-size: 200%;  
    height: 50%;  
    border: 1px solid red;  
}
```

# Jednostki viewport a font-size

## Responsywne czcionki

Jednostki viewport można użyć do ustawienia wielkości czcionki.

Rozmiar tekstu będzie się zmieniał w zależności od wielkości obszaru roboczego (viewportu).

## CSS

```
h2 {  
    font-size: 6vw;  
}  
h3 {  
    font-size: 2vw;  
}
```

# Jednostki viewportu – inne wielkości

## Jednostki relatywne do viewportu

**1vw** = 1% szerokości viewportu;

**1vh** = 1% wysokości viewportu;

**1vmin** = **1vw** lub **1vh**, która wartość jest mniejsza;

**1vmax** = **1vw** lub **1vh**, która wartość jest większa;

Z tych własności wynika, że w przypadku viewportu (tzw. obszaru roboczego) ekranu o szerokości 1000px wartość dla 1vw jest równa:  $1\% \times 1000\text{px} = 10\text{px}$ .

## HTML

```
<div class="parent"></div>
```

## CSS

```
.parent {  
    height: 50vh;  
    width: 50vw;  
    border: 1px solid blue;  
}
```

Element będzie zajmował 50% wysokości i szerokości okna.

# vh vs %

## Jednostki relatywne do viewportu

Jeśli chcielibyśmy ustawić element na wysokość 100% w następujący sposób:

```
.parent {  
    height: 100%;  
}
```

...to doskonale wiesz, że to nie zadziała, dopóki nie ustawimy wysokości dla `html` oraz `body`. Ale element `parent` nie może mieć już żadnego innego przodka:

```
html, body {  
    height: 100%;  
}
```

Można użyć w takiej sytuacji `vh` i nie musimy się martwić o zagnieżdżenie ani ustawianie innym elementom jakieś wysokości.

```
.parent {  
    height: 100vh;  
}
```

# Breakpoints



# Punkty graniczne

**Breakpoint** – punkt graniczny a inaczej szerokość ekranu, przy której zmieniają się obowiązujące style CSS.

Za pomocą punktów granicznych najlepiej jest określić zakres szerokości a nie konkretną szerokość ekranu. Dzięki temu nie trzeba definiować osobnych reguł dla wszystkich możliwych rozdzielczości ekranów i unikniemy luk w szerokościach, dla których nie jest zdefiniowana reguła.

```
@media only screen and (min-width: 600px) { }
```

# Jak dobierać punkty graniczne?

- Dobranie punktów granicznych pod rozmiary używanych urządzeń może okazać się trudne i nieoptymalne ze względu na ich dużą liczbę.
- Dobrą praktyką jest dobranie breakpointów na podstawie treści witryny.
- Najlepiej jest zacząć od najmniejszego ekranu i rozszerzać go do momentu, kiedy przestanie dobrze wyglądać i w tym miejscu ustawić kolejny breakpoint.

## Standardowe breakpointy:

**320px** - iPhone 4

**480px** - Nokia Lumia

**600px** - Inne Phone

**768px** - iPad

> **800px** - Wychodzimy z zakresu pojedynczych kolumn - układ mobile

**1024px** - iPad Landscape

> **1280px** - Desktop, Laptop

**1366px** - Najpopularniejsze laptopy

> **1600px** - HD desktop

# Podsumowanie

- xs - max-width: 575px,
- sm - min-width: 576px,
- md - min-width: 768px,
- lg - min-width: 992px,
- xl - min-width: 1200px,
- hd - min-width: 1600px.

# Jednostki dla punktów granicznych

- Specyfikacja CSS3 mówi, że w `media queries` mogą być stosowane jednostki dostępne w CSS.
- Najczęściej szerokość ta jest określana w pikselach lub `em`-ach.
- Jednostki `em` w `media query` są obliczane według rozmiaru czcionki zdefiniowanego w przeglądarce (nie rozmiaru czcionki, który określmy w CSS dla tagu `html`). W znaczącej większości przypadków domyślnie jest to `16px`.

## Przykład

- Różnica przy stosowaniu `px` i `em` widoczna jest, gdy użytkownik zmieni domyślny rozmiar czcionki.
- Punkty wyliczone w oparciu o jednostki `em` zostaną przesunięte wraz ze zmianą wielkości tekstu.
- W `px` punkty zostaną zachowane, choć wielkość tekstu ulegnie zmianie.



# Wzorce responsywności

# Semantyczny HTML

## Co to jest semantyka?

Semantyczny czyli znaczeniowy kod HTML jest również istotny.

**Semantyka oznacza wykorzystanie tagów HTML zgodnie z ich przeznaczeniem.**

Dzięki semantycie treść strony może być łatwo dostępna również dla urządzeń nieobsługujących zaawansowanych technik CSS i JavaScript.

## Przykład

Jeśli nadamy polu tekstowemu typ email, to wyświetli się użytkownikowi urządzenia mobilnego klawiatura ekranowa z układem, w którym znak „@” jest od razu widoczny:

```
<input type="email"/>
```

# Wzorce responsywnej nawigacji

Wyświetlanie pełnego elementu nawigacji np. w tablecie czy smartfonie byłoby nieoptymalnym rozplanowaniem przestrzeni ekranu.

Ogólnie przyjętą praktyką jest ukrywanie menu np. pod postacią ikony (tzw. **hamburger menu**) i wyświetlanie odnośników na żądanie.

Można też menu wyświetlać np. w formie zwijanej listy.



# Wzorce responsywnej nawigacji

Przykład rozwijanej listy `<select>`, tzw.  
**dropdown menu**

- <http://tinynav.viljamis.com>
- <http://css-tricks.com/convert-menu-to-dropdown>

Przykład **hamburger menu** lub **offcanvas**

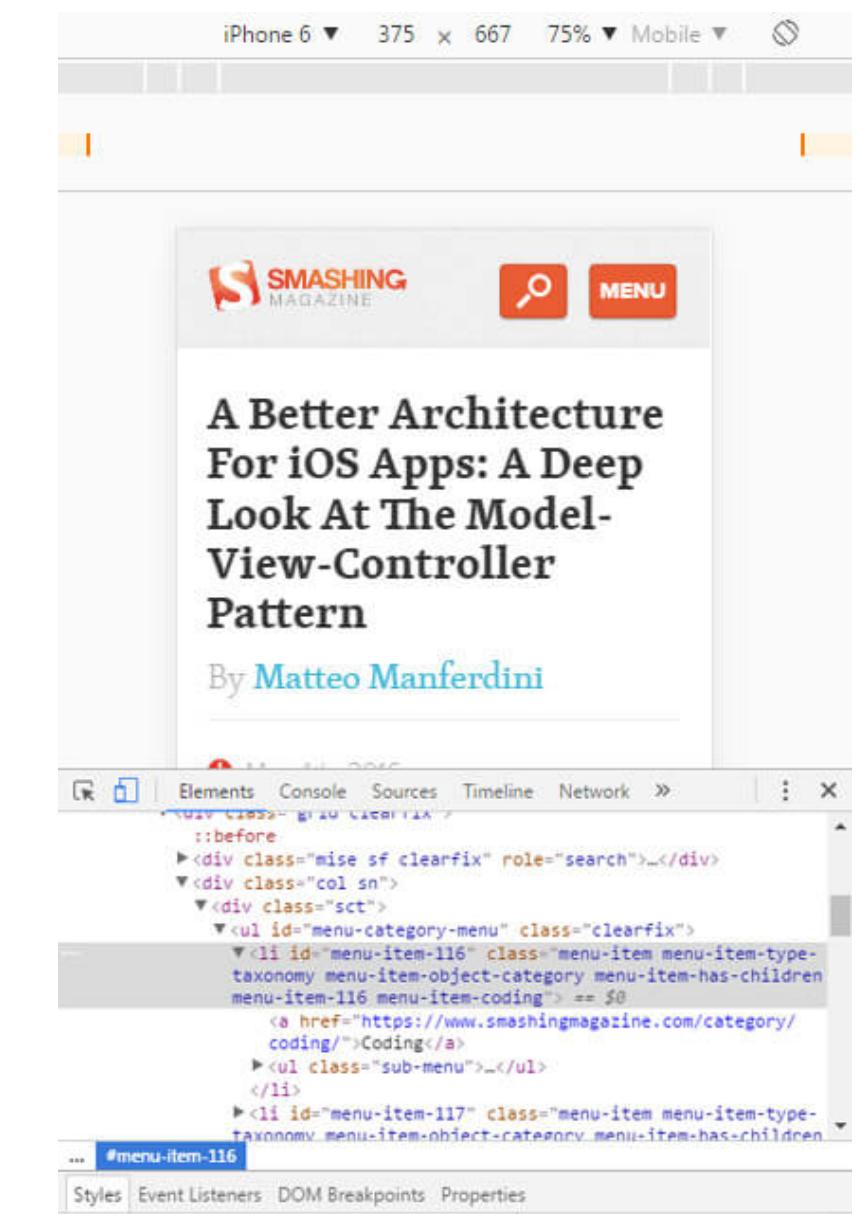
- [http://tympanus.net/Development/OffCanvas\\_MenuEffects/index.html](http://tympanus.net/Development/OffCanvas_MenuEffects/index.html)



# Narzędzia do RWD

# Przydatne narzędzia

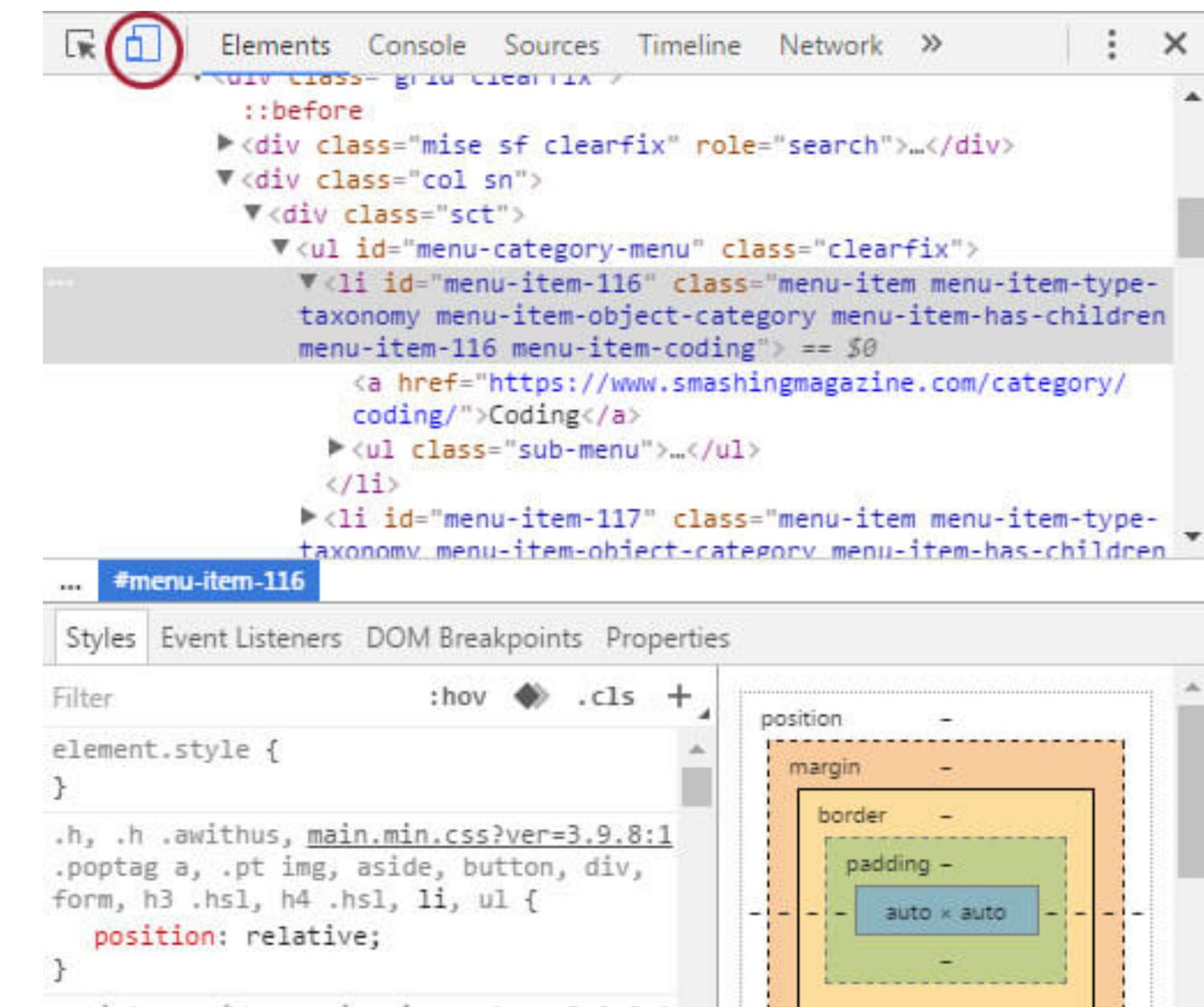
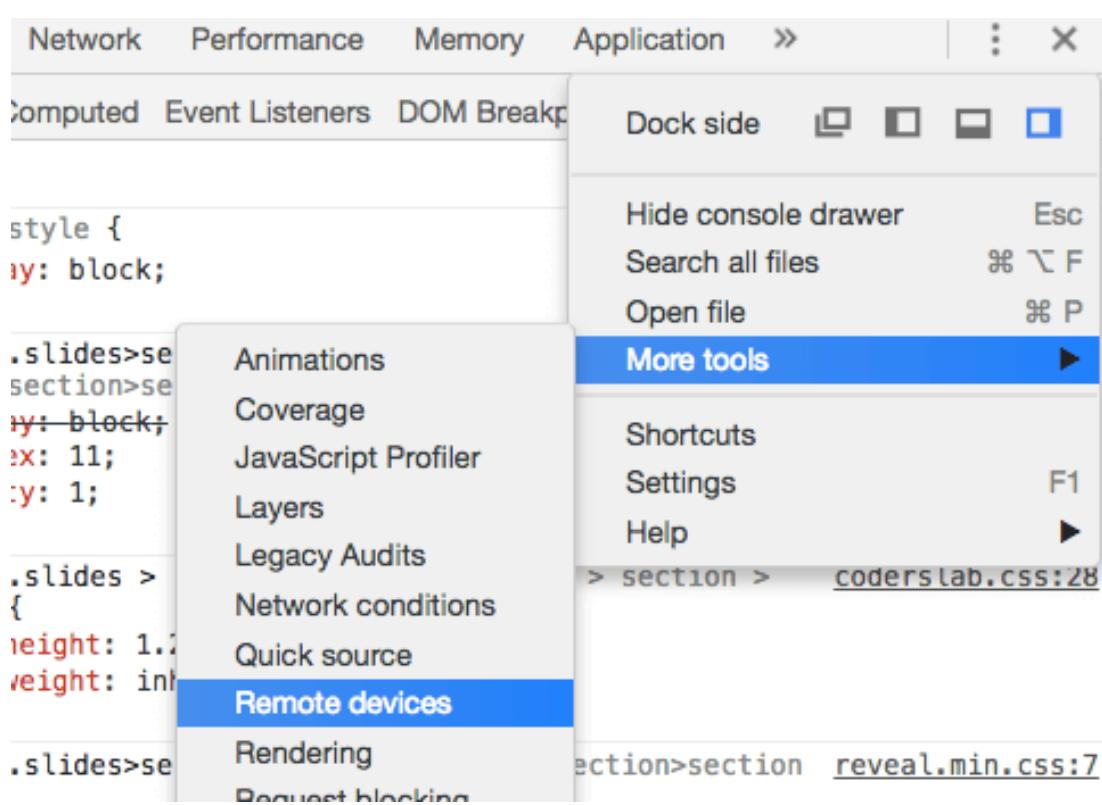
- Responsywność wymaga testowania strony na różnych urządzeniach. Na szczęście nie trzeba mieć dostępu do każdego z nich.
- Nieodłącznym narzędziem pracy każdego programisty są narzędzia deweloperskie wbudowane w przeglądarki.



# Device Mode w Chrome

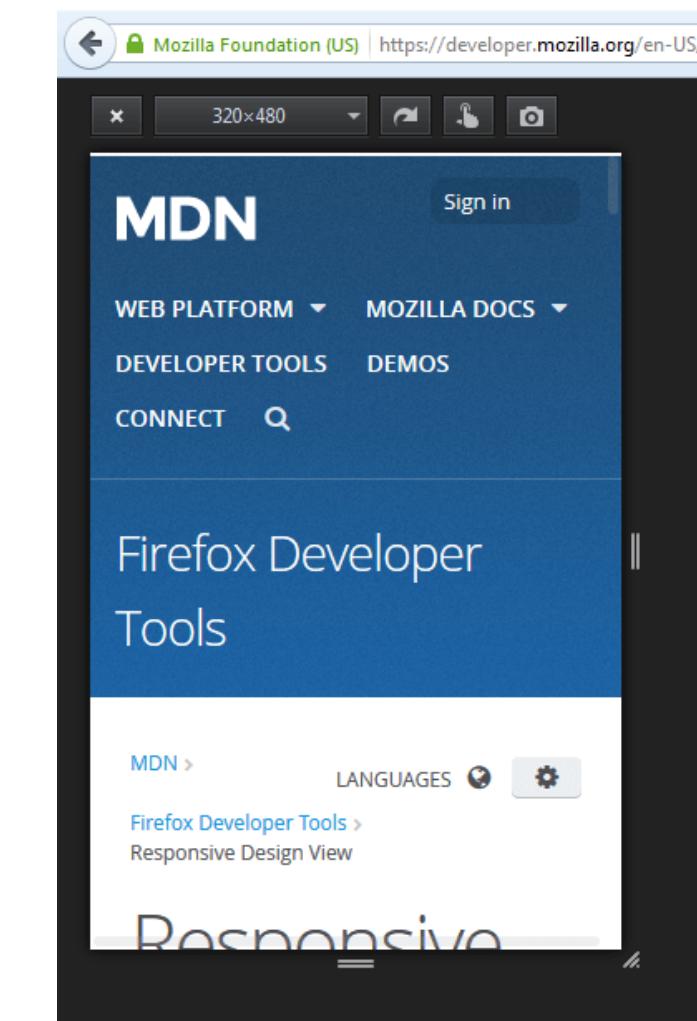
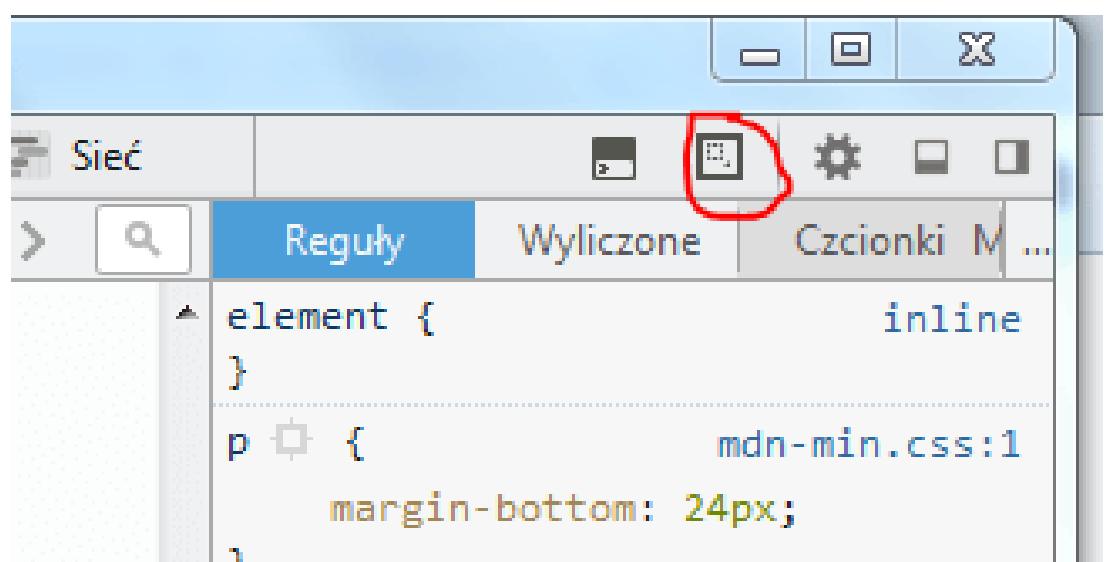
Przeglądarka Chrome ma tryb **Device Mode** pozwalający na emulowanie ekranu różnych urządzeń oraz prędkości połączenia internetowego.

Inna ciekawa funkcjonalność to „Remote devices...” czyli debugowanie strony wyświetlanej na urządzeniu mobilnym podłączonym do komputera.



# Firefox – widok responsywny

Widok ten można włączyć z konsoli narzędzi deweloperskich lub z menu **Narzędzia**.



# Emulacja urządzeń

Apple i Google umożliwiły emulację swoich systemów dla urządzeń mobilnych.

Służą do tego narzędzia **X Code** (dostępny tylko w OS X) oraz **Android Studio** (dostępny na wielu platformach).

Jeśli mamy taką możliwość, to warto sprawdzić stronę na urządzeniu, jako że czasami pojawiają się niespójności między emulatorem a faktycznym urządzeniem np. numer telefonu nie zostanie rozpoznany jako odnośnik.

# Przydatne wskazówki

Pamiętaj o **semantycznym HTML-u**, aby treść była dostępna dla wszystkich.

Korzystaj z podejścia **mobile first**, by kod był optymalny i łatwy w utrzymaniu.

Używaj jednostek względnych jak **%, em, rem** dla łatwego skalowania układu i treści.

**Punkty graniczne** ustawiaj na podstawie treści strony.

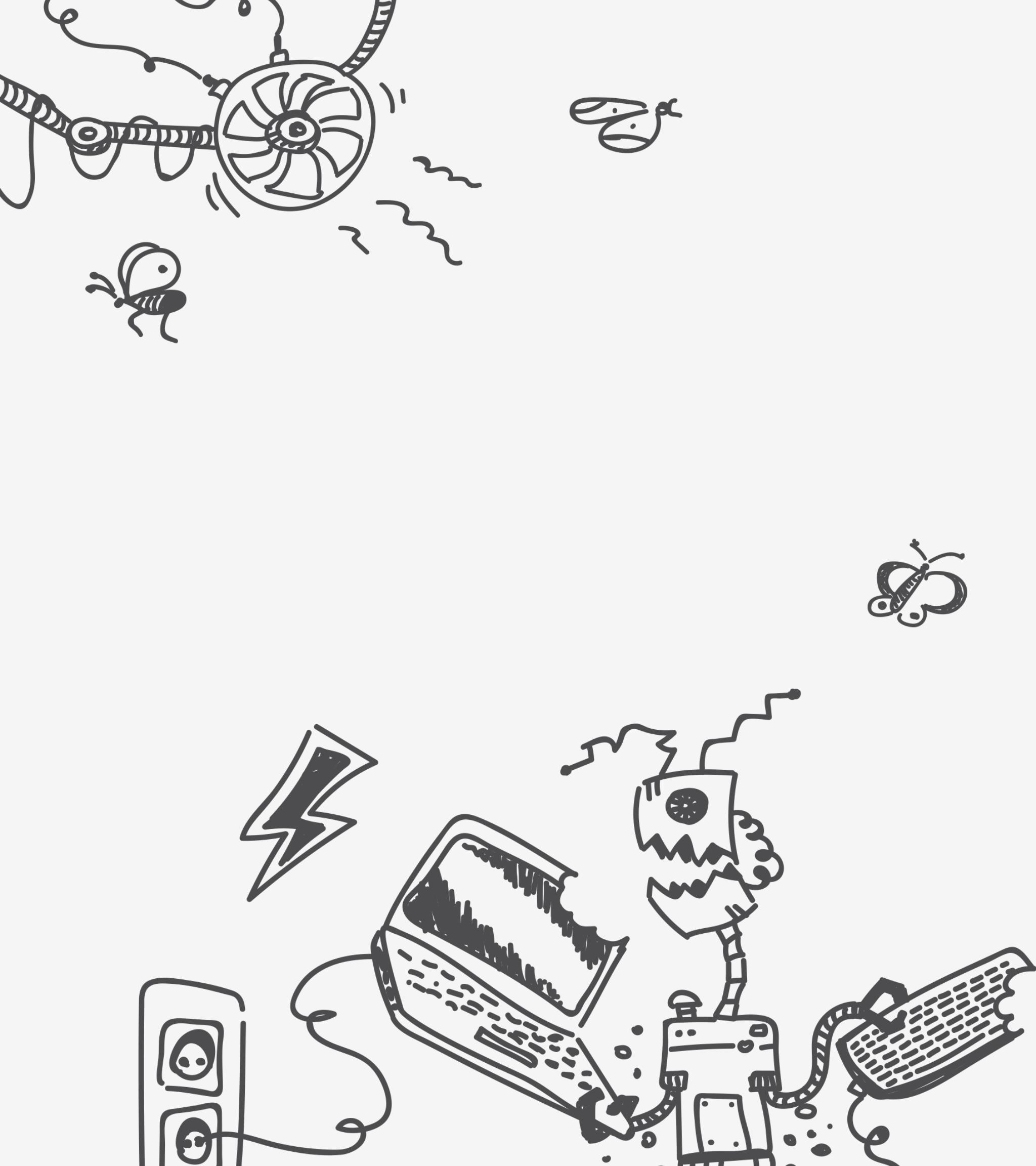
Minimalizuj liczbę zapytań do serwera przez warunkowe wczytywanie treści – CSS zamiast obrazków – a jeśli musisz użyć, zrób to za pomocą **sprites**.

Używaj JavaScript rozsądnie i optymalnie. Użyj **Modernizr**, aby serwować JavaScript tylko do przeglądarek, które są w stanie go wykonać.

Nie ukrywaj zupełnie treści, zamiast tego wyświetl ją pod inną postacią stosowną dla danej wielkości ekranu.

# Przydatne linki

- <http://developers.google.com/web/fundamentals>
- <http://mediaqueri.es>
- <http://www.alistapart.com/articles/responsive-web-design>
- <http://screensiz.es/phone>
- [http://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media\\_queries](http://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries)



# Elastyczne media

1

# Elastyczne media

## Czym jest elastyczność?

Media, czyli w głównej mierze obrazy i wideo, powinny również elastycznie dopasować się do szerokości obszaru, w którym są wyświetlane.

**Najlepiej definiować szerokość mediów przez zastosowanie jednostek względnych wobec ich rodzica.**

```
img, video, canvas {  
    max-width: 100%;  
}
```

# Elastyczne obrazy

## Inna metoda

Można dopasować obraz zdefiniowany w **CSS** jako tło. Służy do tego właściwość **background-size**.

Może ona przyjmować wartości zdefiniowanej szerokości **(px, em, %), auto, cover, contain, initial, inherit**.

```
img, video, canvas {  
    max-width: 100%;  
}  
.bg-cover {  
    width: 100%;  
    height: 100vh;  
    background-image: url("image.png");  
    background-size: cover; /* contain, 100% auto */  
    background-repeat: no-repeat;  
    background-color: transparent;  
}
```

# Optymalizacja obrazów

## Jak zadbać o optymalizację?

Dobrą praktyką jest optymalizacja rozmiaru i jakości obrazów wyświetlanych na stronach.

Pozwala to na szybsze załadowanie strony oraz oszczędności transferu, które są istotne zwłaszcza na łączach sieci komórkowej.

Warto podjąć próby dostosowania jakości do poszczególnych obrazów, gdyż efekt nie zawsze jest jednakowy.

Przydatny plugin do Gulp – optymalizacja obrazów: <https://github.com/sindresorhus/gulp-imagemin>

Program do bezstratnej kompresji obrazów: <https://imageoptim.com/pl>

Strony do optymalizacji:

- <https://tinypng.com/>
- <https://compressor.io/compress>
- <https://squoosh.app>

# CSS Sprites

## PNG

**Portable Network Graphics** to format zalecany przez **w3c** do stosowania grafiki na stronach internetowych. Charakteryzuje się bezstratną kompresją obrazu.

Przydaje się do wyświetlania obrazów o niewielkim rozmiarze a zwłaszcza tych, które wymagają częściowej przezroczystości.

W porównaniu do innych formatów rozmiar takiego obrazu jest z reguły najmniejszy przy zachowaniu wysokiej jakości.

# CSS Sprites

## Minimalizowanie zapytania

Aby zminimalizować zapytania wysyłane do serwera (`HTTP Requests`) obrazy można zapisać w jednym pliku i odwoływać się do nich, przesuwając odpowiednio tło.

Flagi widoczne po prawej stronie są jednym plikiem PNG.

Przydatny plugin generujący kod z pozycjami elementów:

<https://github.com/twolfson/gulp.spritesmith>



# CSS Sprites

## Umieszczanie CSS Sprites

Aby umieścić obrazy (w tym przypadku – flagi) w trzech różnych elementach strony, można użyć następującego kodu CSS.

```
.flags-canada, .flags-mexico, .flags-usa {  
background-image: url("../images/flags.png");  
background-repeat: no-repeat;  
}
```

```
.flags-canada {  
height: 128px;  
background-position: -5px -5px;  
}  
.flags-usa {  
height: 135px;  
background-position: -5px -143px;  
}
```

```
.flags-mexico {  
height: 128px;  
background-position: -5px -288px;  
}
```

# Responsywne obrazy CSS

## Jak optymalizować?

Obrazy wstawione za pomocą CSS również można optymalizować dla różnego typu mediów.

Dla małych ekranów serwujemy obraz o niskiej rozdzielczości, a dla dużych – o wysokiej, ale aby ich wielkość nie przekraczała górnej wartości punktu granicznego.

```
.example {  
    height: 400px;  
    background-repeat: no-repeat;  
    background-size: contain;  
}  
  
@media (max-width: 499px) {  
    .example {  
        background-image: url("small.png");  
    }  
}  
  
@media (min-width: 500px) {  
    .example {  
        background-image: url("large.png");  
    }  
}
```

# Responsywne obrazy CSS

## Ekrany Retina

Podobnie możemy wykorzystać **media query** dla ekranów z większą gęstością pikseli (Retina).

```
.example {  
    background-image: url('ico.png');  
}  
  
@media only screen and (-webkit-min-device-pixel-ratio: 2),  
      only screen and (min-resolution: 2dppx) {  
    .example {  
        background-image: url('ico@2x.png');  
    }  
}
```

# Responsywne obrazy

Innym wyzwaniem jest serwowanie odpowiedniego pliku obrazu umieszczonego w tagu HTML tak, aby był sformatowany i zoptymalizowany dla danego typu urządzeniu lub ekranu.

Trudność polega na braku łatwej i ustandaryzowanej metody.

# Metoda srcset

## Co to takiego?

Jest to jedna z proponowanych przez W3C metod serwowania innego źródła dla danego tagu `<img>` w zależności od ekranu.

Warunkiem może być gęstość pikseli lub szerokość, jaką na ekranie będzie miał obraz.

Jest to wciąż robocza wersja HTML, więc w wersji końcowej mogą pojawić się zmiany.

Dlatego też ta metoda nie została wdrożona we wszystkich przeglądarkach.

<http://www.w3.org/html/wg/drafts/html/master/semantics.html#attr-img-srcset>

```

```

- Przykład zastosowania atrybutu **srcset**  
<http://webkit.org/demos/srcset>

# Metoda picture

Metoda ta pozwala na użycie odpowiedniego źródła dla danego **media query**.

```
<picture>
  <source media="(min-width: 45em)" srcset="large.jpg">
  <source media="(min-width: 32em)" srcset="med.jpg">
  
</picture>
```

# Polyfill

- Nie wszystkie przeglądarki obsługują `picture`,  
`srcset` czy też `sizes`.
- Możemy więc podłączyć skrypt, który pozwoli na obsługę tych metod. Jest to tzw. `polyfill`. Jednym z nich jest [Picturefill](#).

# Ikony wektorowe

## Deklaracja @font-face

Popularnym rozwiązaniem dostosowania ikon do różnych ekranów jest użycie ich w postaci czcionki i deklaracji `@font-face`.

Dzięki temu ikony są zdefiniowane za pomocą wektorów i mogą być skalowane do różnych rozmiarów bez strat na jakości.

Ważną częścią podczas tworzenia projektu jest odpowiednie radzenie sobie z czcionkami np. svg użytymi w projekcie. Możemy użyć wtyczki do Gulpa <https://github.com/jvanaert/gulp-webfont> lub Icomoon <https://icomoon.io>.

## Gotowe i darmowe czcionki ikon

- FontAwesome,
- Fontello (generator),
- Genericons,
- Entypo.

Ikon wektorowych używa się ich przez użycie klas, np:

```
<i class="fa fa-globe"></i>
```

# Skalowanie zagnieżdżonego wideo

Największą trudnością przy skalowaniu zagnieżdżonego wideo (np. w tagu `iframe`) jest zachowanie jego proporcji.

W przeciwieństwie do obrazków, wysokość elementu `iframe` musi być zdefiniowana i nie jest skalowana proporcjonalnie do jego szerokości.

Jest jednak na to sposób.

# Skalowanie zagnieżdżonego wideo

## Kod HTML

```
<figure>
  <iframe src="https://www.youtube.com/embed/4Fqg43ozz7A">
  </iframe>
</figure>
```

## Kod CSS

```
figure {
  height: 0;
  padding-bottom: 56.25%; /* 16:9 */
  position: relative;
  width: 100%;
}
```

```
iframe {
  height: 100%;
  left: 0;
  position: absolute;
  top: 0;
  width: 100%;
}
```

# Typografia

## Punkt graniczny

Zakłada się, że przyszłość mobile opiera się głównie na typografii.

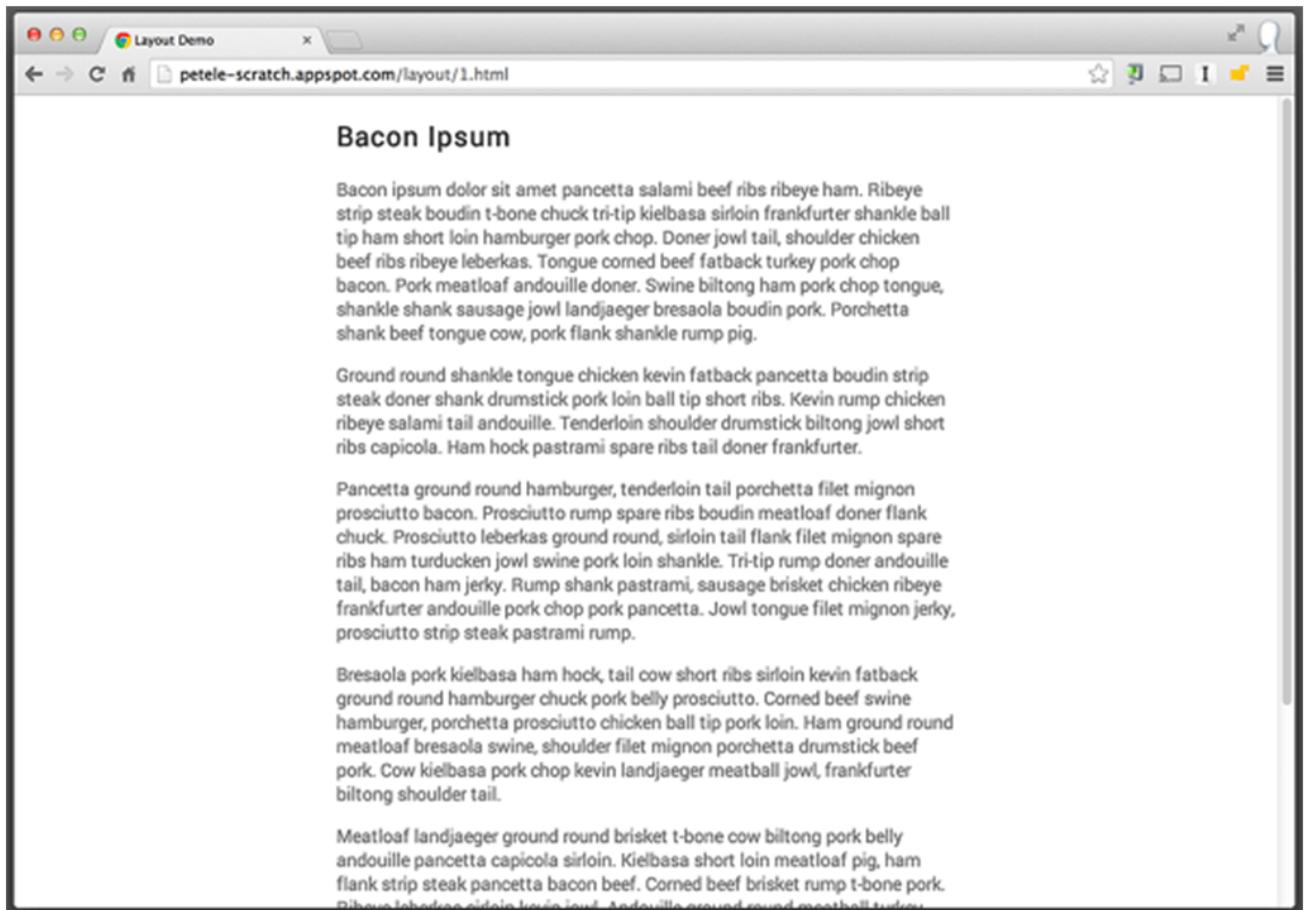
Mało dostępnego miejsca wymusza redukcję liczby informacji widocznych na ekranie.

Czytelność tekstu staje się zatem kluczowa.

Według klasycznych zasad gwarantujących czytelność tekstu, idealna szpalta powinna zawierać 70–80 znaków w wierszu (około 8–10 wyrazów).

Za każdym razem, gdy wiersz w bloku tekstu przekroczy 10 wyrazów, należy rozważyć utworzenie punktu granicznego.

# Responsywna typografia



# Responsywna typografia

- Responsywna typografia powinna opierać się na jednostkach względnych.
- Taki tekst łatwiej skalować i dostosowywać do preferencji użytkownika.

```
html { font-size: 62.5%; }
h1 { font-size: 10vw; }
h2 { font-size: 2.2rem; /* 22px */ }
h3 { font-size: 1.8rem; /* 18px */ }
```

```
@media (min-width: 480px) {
    h2 { font-size: 5rem; /* 50px */ }
    h3 { font-size: 3rem; /* 30px */ }
}
```

```
@media (min-width: 1024px) {
    h2 { font-size: 8rem; /* 80px */ }
    h3 { font-size: 5rem; /* 50px */ }
}
```