

# Responsive Web Design



Daniel Bednarek

# Organizacja zajęć

- 2 przerwy po 10 minut
- W trakcie zajęć zgłaszamy, gdy utkniemy
- Nie ma głupich pytań
- Do zadań używamy plików z Dysku Google z folderu DB
- Każde zadanie posiada folder *starter* w którym wykonujecie zadanie
- Folderu *final* staramy się nie podglądać, zawiera on gotowe rozwiązanie
- Plik *zrodla.md* zawiera źródła przydatne do używania w trakcie zajęć i nauki po zajęciach
- Otwieramy folder *RWD* przez VS Code i wykonujemy zadania nawigując w folderach

# Narzędzia

- Wyszukiwarka Google
- VS Code
- Materiały z Dysku Google
- Przeglądarka Chrome

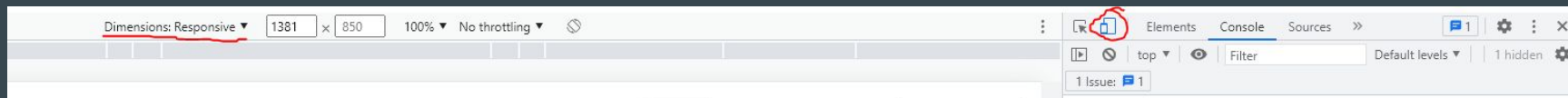
# Zakres zajęć

1. Czym jest RWD? Czemu jest on tak ważny?
2. <head> - viewport metadata
3. Porównanie jednostek i ich wpływu na responsywność
4. Responsywne obrazy
5. Layout Strony - wprowadzenie
6. Tabele
7. Breakpoints
8. CSS Media Queries
9. Tag <picture>
10. CSS Flexbox
11. CSS Grid
12. CSS Multicol
13. Bootstrap

# Czym jest RWD?

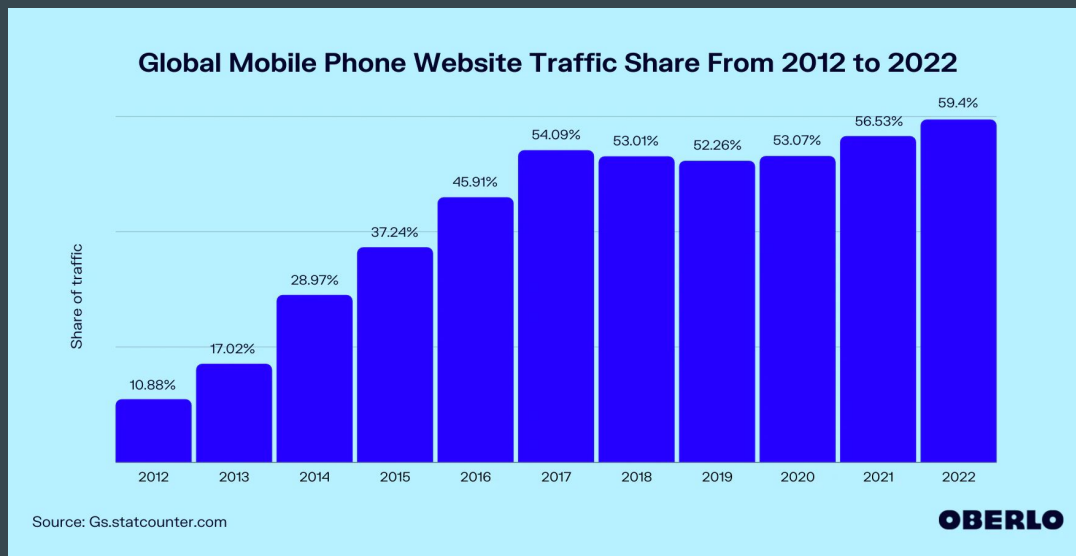
- Responsive Web Design to sposób tworzenia stron internetowych tak, by dostosowane były do każdego urządzenia, które może przeglądać internet
- Responsywna strona internetowa reaguje na zmiany rozmiaru ekranu i jest wygodna w użytkowaniu na każdym urządzeniu

Zadanie 1 - Otwórz link 1.a z pliku zrodla.md. Włącz DevTools (F12), włącz Device Toolbar i wybierz Dimensions: Responsive. Zmniejszaj i zwiększaj szerokość ekranu. Jak zachowuje się strona?

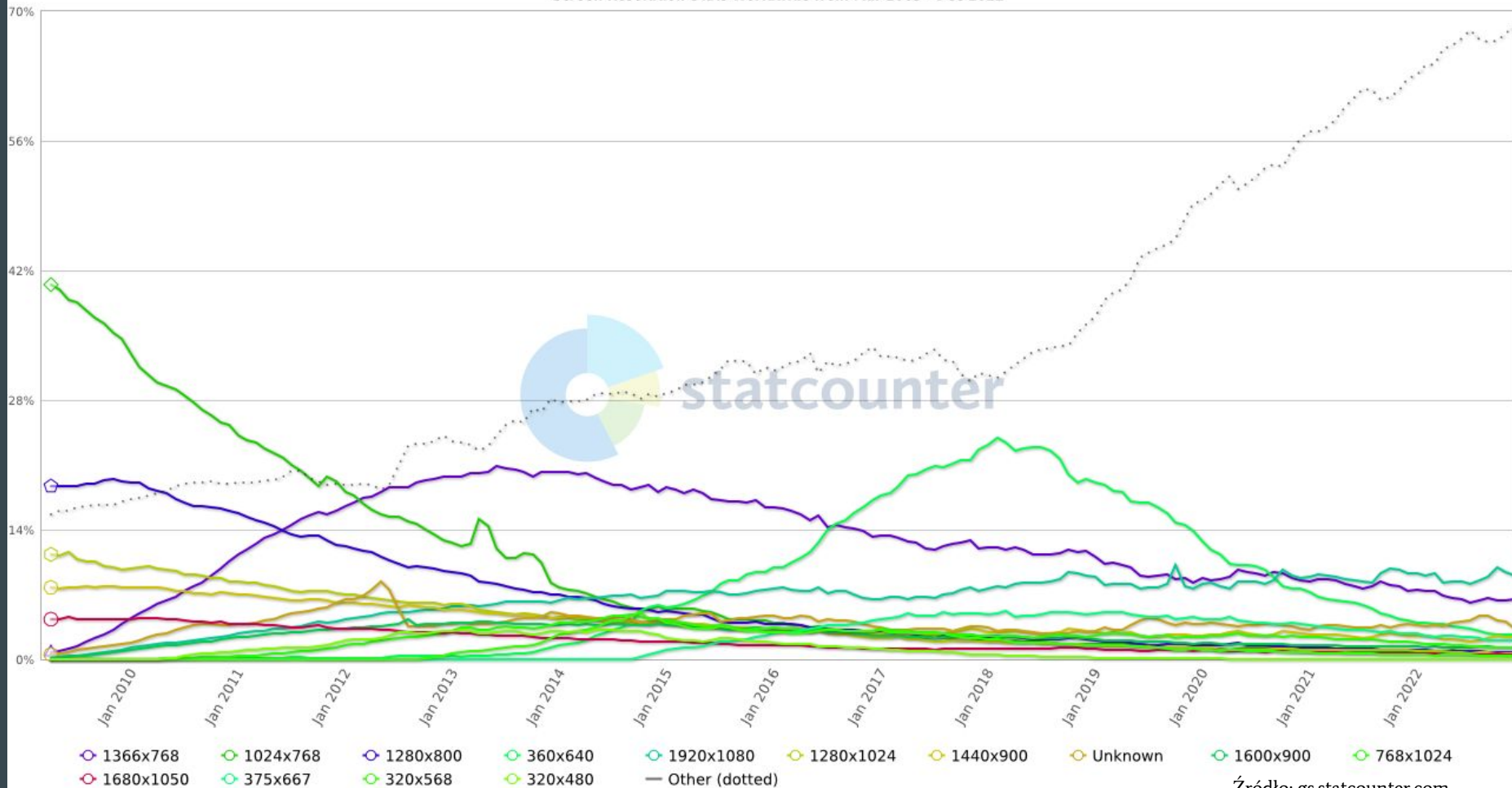


# Czemu RWD jest ważny?

- Coraz więcej urządzeń ma możliwość przeglądania internetu: laptopy, telefony, TV, tablety, a nawet smartwatche
- Nasza strona musi być przygotowana na różne rozdzielczości, kształty i rozmiary ekranów



StatCounter Global Stats  
Screen Resolution Stats Worldwide from Mar 2009 - Dec 2022



# Viewport metadata

- Konieczne, by nasza strona była responsywna
- Instrukcja dla przeglądarki, jak ma wyrenderować stronę i dostosować jej rozmiar do ekranu urządzenia
- Dodawane automatycznie podczas tworzenia nowego pliku .html w **VS Code** i napisaniu na początku wykrzyknika i wybranie pierwszej opcji

Zadanie 1 - utwórz plik .html w VS code i wpisz wykrzyknik. Wybierz pierwszą opcję

```
<head>
  <meta charset="UTF-8">
  <!-- -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- -->
  <title>Document</title>
</head>
```



# Jednostki w CSS

- Używane do określania rozmiarów elementów
- Dzieli się na **absolutne** i **relatywne**

## absolutne

cm

centimeters

mm

millimeters

in

inches (1in = 96px = 2.54cm)

px \*

pixels (1px = 1/96th of 1in)

pt

points (1pt = 1/72 of 1in)

pc

picas (1pc = 12 pt)

## relatywne

em



Relative to the font-size of the element (2em means 2 times the size of the current font)

ex

Relative to the x-height of the current font (rarely used)

ch

Relative to the width of the "0" (zero)

rem



Relative to font-size of the root element

vw



Relative to 1% of the width of the viewport\*

vh



Relative to 1% of the height of the viewport\*

vmin

Relative to 1% of viewport's\* smaller dimension

vmax

Relative to 1% of viewport's\* larger dimension

%



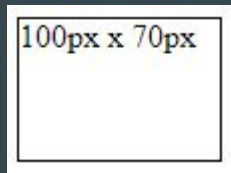
Relative to the parent element

\* Pixels (px) are relative to the viewing device.

# Pixels (px)

- Używane do ustawiania rozmiarów elementów “na sztywno”
- Przydatne, gdy chcemy, by element miał zawsze takie same rozmiary
- Najczęściej używana jednostka absolutna
- Urządzenia z ekranami o większej gęstości pikseli (np. Retina) automatycznie dostosowują się do “standardu” mniejszej gęstości

```
<body>  
  <div> 100px x 70px</div>  
</body>  
  
<style>  
  div {  
    width: 100px;  
    height: 70px;  
    border: 1px solid black;  
  }  
</style>
```



# Procent (%)

- Najprostsza jednostka relatywna
- Procenty odnoszą się do rozmiaru **rodzica** stylowanego elementu
- Element o wysokości i szerokości wyrażonej w procentach sam dostosowuje się do rozmiaru ekranu
- Nie jest dobrym pomysłem używanie procentów jako rozmiaru niektórych właściwości, np. paddingu i marginu elementu. Prowadzi to do trudnych do przewidzenia, lub nie intuicyjnych zachowań (źródło 3.c)

Zadanie: Otwórz plik example2.html w przeglądarce i w VS Code. Zmieniaj rozmiar okna przeglądarki i zobacz jak zachowają się elementy. Przeczytaj kod pliku w VS Code.

# Em (em)

- Jednostka relatywna
- Używana do określania rozmiaru czcionki
- Przejmuje rozmiar czcionki rodzica i mnoży przez dodaną wartość
- Na przykład, gdy rodzic ma rozmiar czcionki ustawiony na 16px, 1em == 16px, 0.5em == 8px, 2em == 32px i tak dalej

Zadanie: Otwórz plik example3.html i sprawdź jak działa jednostka *em* na własne oczy! Nie bój się zmieniać wartości i eksperymentować.

# Root Em (rem)

- Jednostka relatywna
- Tak jak *em*, używane do określania rozmiaru czcionki
- Odnosi się do rozmiaru czcionki root-u strony internetowej, czyli elementu `<html>`
- Domyślny rozmiar czcionki dla tagu `<html>` w przeglądarkach to 16px, czyli w tym wypadku `1rem == 16px`, `2rem == 32px` itd.
- Jeżeli użytkownik zmieni domyślny rozmiar czcionki dla przeglądarki, jednostki *rem* zareagują odpowiednio i pomnożą domyślny rozmiar przez dane im wartości

# View Width (vw)

- Jednostka relatywna
- 1vw to odpowiednik 1% szerokości okna przeglądarki
- Jednostki % i vw to nie są odpowiedniki! % może być procentem czegokolwiek, a vw to procent szerokości okna (źródło 3.d)
- Można na przykład ustawić wysokość elementu na 10vw, więc wysokość elementu będzie równała się 10% szerokości okna przeglądarki



# View Height (vh)

- Jednostka relatywna
  - 1vh to odpowiednik 1% wysokości okna przeglądarki
  - Podobnie jak vw, jednostka vh nie jest odpowiednikiem jednostki %
- 
- Reszta jednostek używana jest rzadko bądź wcale. Użyj źródeł, oraz Google, by dowiedzieć się o nich więcej

# Zadanie 1

Otwórz Zad\_1/starter/starter.html i edytuj CSS w tagu <style>:

- <div> z klasą *class1* ustaw na pełną szerokość okna i 200 pikseli wysokości. Rozmiar fontu ustaw na podwójną wartość rozmiaru fontu **rootu strony**.
- <div>-y z klasami *class2* i *class3* ustaw każdy na 40% szerokości strony i 200 pikseli wysokości. Rozmiar fontu ustaw na 1.5 rozmiaru fontu **rodzica**.
- Dodaj bordery do divów, by było widać je na stronie

Po sprawdzeniu Zadania 1 wspólnie: **Zadanie 1.1:**

- Usuń ten element kodu CSS i postaraj się wytłumaczyć co się stało:

```
div {  
  display: inline-block;  
}
```



# Responsywne obrazy

- Odpowiednie ustawienie wysokości i szerokości obrazów zapobiega takim sytuacjom:



- Jeżeli chcemy by obrazy skalowały się w górę i w dół, musimy dodać styl *width=100%* i *height=auto* w CSS
- Może to mieć negatywne konsekwencje, np. obraz rozciągnie się poza swoją rozdzielczość i będzie rozmazany
- Lepszą alternatywą jest ustawienie *max-width=100%* i *height=auto* w CSS
- Dzięki temu obraz będzie skalował się w dół i w górę, ale nigdy powyżej swojego oryginalnego rozmiaru

Zadanie: Otwórz plik `example1.html` i sprawdź jak wygląda kod CSS i jak zachowują się obrazy w przeglądarce

# Layout Strony - wprowadzenie

- Elementy na stronie mają swoje domyślne pozycje i zachowania, które możemy modyfikować dzięki różnym technikom i narzędziom
- Sprawdź plik `example1.html` w którym są 2 listy o takiej samej zawartości i porównaj stylujące je fragmenty CSS. Semantycznie listy są praktycznie identyczne.
- Dla użytkownika są one wizualnie różne, ale boty i czytniki ekranu (na przykład takie, które pozwalają osobom niewidomym “czytać” strony) nadal widzą je w postaci semantycznej.
- Odpowiednie dobranie technik modyfikowania layoutu jest kluczowe dla dostępności naszej strony internetowej
- Dostępność jest także ważna dla botów Googla, które na jej podstawie mogą umieścić naszą stronę wyżej, lub niżej w wynikach wyszukiwania
- Pamiętaj, żeby zmiany wizualne zawsze miały sens, gdy porówna się je z kodem strony!

# Podstawowe techniki zarządzania layoutem

- Właściwość *display* może modyfikować layout i cechy elementu. Na przykład możemy zmienić domyślną wartość dla tagu `<div>` z *display: block*, nadpisując ją za pomocą kodu CSS na *display: inline-block*, by dany `<div>` przestał domyślnie zabierać całą szerokość okna przeglądarki
- Wartości *display*, to *inline*, *flex*, *grid*, *block*, *inline-block*, *none* (usuwa element z warstwy wizualnej, ale zostawia go w kodzie strony)
- Właściwość *position* kontroluje jak element zachowuje się wizualnie i wobec innych elementów strony. Możliwe wartości to *static*, *relative*, *absolute*, *fixed*, *sticky*
- *z-index*: określa który z nakładających się na siebie elementów jest “na górze”.

Powinieneś już znać podstawy wyżej wymienionych technik, ale w razie czego możesz wspomóc się źródłami 5.a, b, c, d

# Szybka powtórka - display:

- block - tworzy blok, który jest jedynym blokiem w swoim wierszu strony
- inline-block - tworzy blok, który jeżeli jest miejsce, nie będzie jedynym blokiem w linii
- none - element istnieje w kodzie strony, ale jest niewidoczny i nie wpływa na layout
- flex i grid - poznamy dziś w kolejnych rozdziałach

# Szybka powtórka - position:

- static - wartość domyślna. Element pozycjonuje się względem innych elementów strony. Z-index nie działa tylko dla tego elementu\*.
- relative - element pozostaje w domyślnej pozycji, ale możemy przesuwać go relatywnie do jego oryginalnego położenia (np *top: 5px* przesunie go o 5px w dół). Uwaga! W początkowym miejscu pozostanie “dziura” po tym elemencie
- absolute - element jest usunięty ze swojej pozycji i nie jest rezerwowane dla niego miejsce. Pojawia się relatywnie do swojego najbliższego rodzica/przodka, który ma określone position, lub zawierającego go bloku. Także można go przesuwać.
- fixed - element zostaje usunięty z pozycji i “przyklejony” do jednego miejsca na ekranie. Można go przesuwać
- sticky - element ma swoją pozycję i przykleja się do góry scrollowanego elementu, który jest jego rodzicem/przodkiem

# Tabele

- Wykorzystywane były do tworzenia layoutów stron, zanim pojawiły się narzędzia takie jak flexbox i grid
- Nadal wykorzystywane są do tworzenia layoutów e-maili (programy pocztowe są często mocno w tyle z obsługą nowych funkcji HTML i CSS)
- Podstawowe elementy tabel to: `<table>` (cała tabela), `<tr>` (table row - wiersz tabeli), `<th>` (table header - wiersz tytułowy), `<td>` (table data - pojedyncza komórka tabeli)

TH	TH	TH	TH
TD	TD	TD	TD
TD	TD	TD	TD



TR

```
<table>
  <tr>
    <th>TH</th>
    <th>TH</th>
    <th>TH</th>
    <th>TH</th>
  </tr>
  <tr>
    <td>TD</td>
    <td>TD</td>
    <td>TD</td>
    <td>TD</td>
  </tr>
  <tr>
    <td>TD</td>
    <td>TD</td>
    <td>TD</td>
    <td>TD</td>
  </tr>
</table>
```

# Zadanie 1

Posługując się przykładem z `example1.html` i źródłami 6.a i 6.b, stwórz tabelę o następujących właściwościach:

- 3 kolumny
- 3 wiersze (w tym 1 tytułowy)
- Zawartość dowolna (może być *Lorem Ipsum*)
- Cała tabela szeroka na 500px;
- Pierwsza kolumna szeroka na 200px;
- Wiersz tytułowy pogrubiony border, mocno szare tło i biały kolor tekstu
- Reszta tabeli border 1px, tło delikatnie żółte, tekst czarny
- Padding w każdej komórce 3px




# Breakpoints

- Breakpoints to wartości szerokości strony, wyrażone w pikselach, które powodują jakąś zmianę stylu, lub layoutu strony
- Używane do budowania responsywnych stron, w przypadkach, gdy strona musi wyglądać inaczej na różnych rozmiarach ekranów
- Liczbę breakpointów i wartość określa deweloper, zależnie od wyglądu strony.
- Zazwyczaj stosuje się breakpointsy dla telefonów, tabletów, oraz komputerów. Można uwzględnić większe ekrany, jak na przykład smart TV, jeżeli nasza strona wygląda źle na tak dużych urządzeniach
- Przykładowe wartości breakpointów to: 360px+ (telefon), 560px+ (tablet), 770px+ (komputer)


# CSS Media Queries

- Pozwalają na zmianę stylowania na podstawie wykrytego typu urządzenia i jego cech
- Jest wiele możliwości użycia Media Queries, przeczytasz o wszystkich w źródłach 8.a i 8.b
- Najpierw użyjemy Media Query reagującego na szerokość ekranu urządzenia



```
@media only screen and (max-width: 500px) {  
  body {  
    background-color: #ff8484;  
  }  
}
```

Styl aplikowany będzie od 0px do 500px

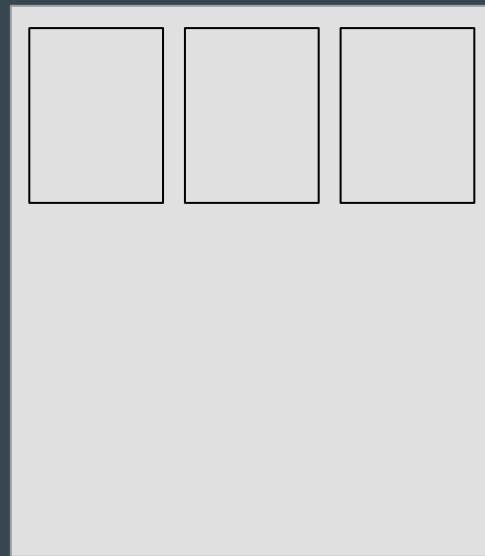
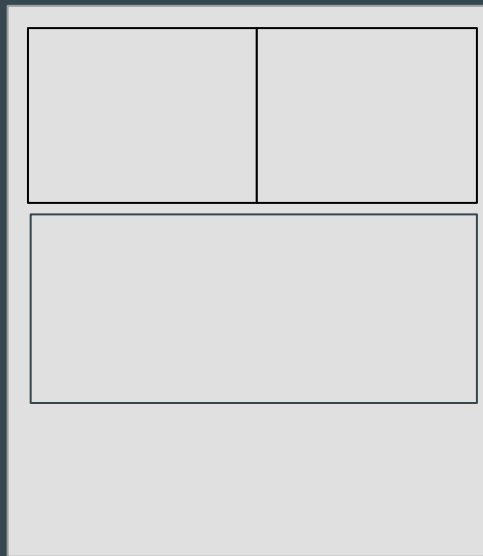
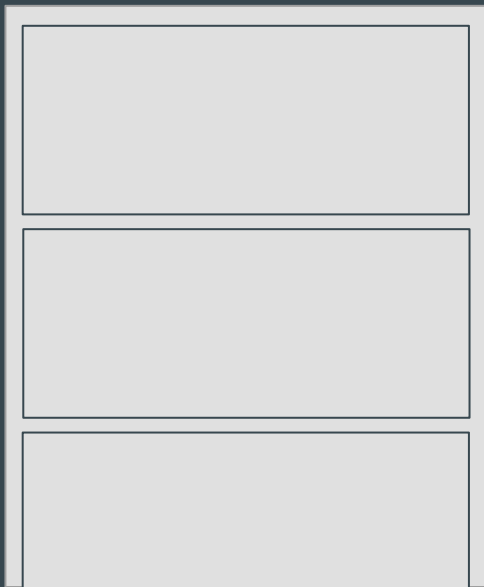


```
@media only screen and (min-width: 700px) {  
  div {  
    display: inline-block;  
    width: 40%;  
  }  
}
```

Styl aplikowany będzie od 700px do nieskończoności

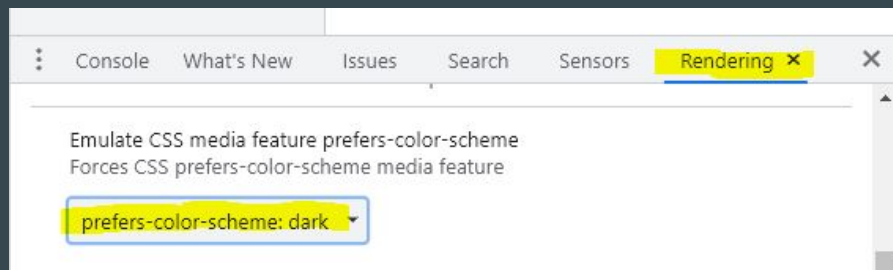
# Zadanie 1

- Przyjmij 3 breakpointy: dla urządzeń mobilnych(max 400px), tabletów(max 600px), oraz komputerów (601px+)
- Nie edytuj kodu HTML, tylko CSS pomiędzy tagami <style>
- W pliku starter.html edytuj CSS, by odwzorować poniższy layout



Prefers-color-scheme - Media Query reagujące na preferencje urządzenia odnośnie trybu ciemnego lub jasnego

- Przyjmuje wartość “dark”, lub “light”
- Można emulować preferencje w przeglądarce w Dev Tools



```
@media (prefers-color-scheme: dark) {  
  body {  
    background-color: #5a5a5a;  
  }  
  
  div {  
    background-color: #000;  
    color: #fff;  
  }  
}
```

example3.html

## Zadanie 2

Wykorzystaj wiedzę o Media Query prefers-color-scheme i dorób tryb ciemny do rozwiązania Zadania 1. Emuluj preferencje trybu ciemnego w DevTools i sprawdź czy Twój kod działa. Możesz też zmienić preferencje swojego urządzenia (działa na iOS, Windows, oraz Android).

Podpowiedź: Nie musisz definiować Media Query dla “light” i “dark”. Wystarczą style domyślne + 1 Media Query nadpisujące je.

# Tag <picture>

- Tag ten pozwala decydować przeglądarce, którego źródła obrazu użyć
- Można zdefiniować kilka źródeł obrazu, dla kilku szerokości ekranu
- Dzięki tej funkcjonalności, użytkownicy np. telefonów nie będą ściągać obrazka w oryginalnym rozmiarze, tylko ten przeznaczony dla mniejszych ekranów
- Można także dodać ten sam obraz w kilku formatach i przeglądarka wybierze pierwszy obsługiwany
- Tag <img> jest obowiązkowy. To jego zaznaczamy do stylowania. Tag <picture> będzie tylko odpowiednio “podmieniał” w nim wartość atrybutu “src”, biorąc ją z tagu <source>

```
<picture>  
  <source media="(min-width: 650px)" srcset="img_food.jpg">  
  <source media="(min-width: 465px)" srcset="img_car.jpg">  
    
</picture>
```

# CSS Flexbox

- Narzędzie do wygodnego tworzenia layoutów dostosowujących się do ekranu
- Używane raczej do layoutów elementów, niż całej strony
- *Flexbox* oznacza Flexible Box, czyli elastyczne pudełko
- Element zdefiniowany jako Flexbox zarządza rozmiarami i rozmieszczeniem elementów znajdujących się wewnątrz niego
- Możemy kontrolować właściwości rodzica oraz dzieci
- Aby zadeklarować element jako Flexbox, trzeba zmienić jego wartość *display* na *flex* (example1.html)

```
.flex-container {  
  display: flex;  
}
```

# CSS Flexbox - wybrane właściwości (źródło 10.a)

flex-direction - kierunek ułożenia elementów w pionie i poziomie

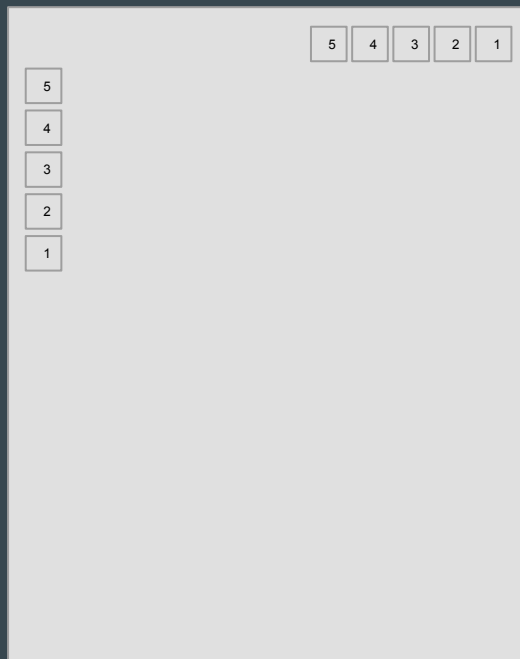
- *flex-direction: row;* → (domyślne) wiersz, od lewej do prawej
- *flex-direction: row-reverse;* → wiersz, od prawej do lewej
- *flex-direction: column;* → kolumna, od góry do dołu
- *flex-direction: column-reverse;* → kolumna, od dołu do góry

```
.flex-container {  
  display: flex;  
  flex-direction: column;  
}
```



# Zadanie 1

Otwórz plik Zad\_1/starter.html i edytuj CSS w tagu `<style>`, by uzyskać poniższy efekt:



flex-wrap - kontrola przeskakiwania elementów do kolejnej linii

- *flex-wrap: nowrap;* → (domyślne) wszystkie elementy w jednej linii
- *flex-wrap: wrap;* → elementy będą przeskakiwać na kolejne linie gdy skończy się im miejsce
- *flex-wrap: wrap-reverse;* → jak wyżej, tylko od dołu do góry (example2.html)

```
.flex-container {  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap-reverse;  
}
```

justify-content - rozmieszczenie elementów wzdłuż głównej osi



- `justify-content: flex-start;` —→ (domyślne) elementy upakowane jak najbliżej początku
- `justify-content: flex-end;` —→ upakowane jak najbliżej końca
- `justify-content: center;` —→ wycentrowane wzdłuż osi
- `justify-content: space-between;` —→ równo rozmieszczone, od brzegów rodzica
- `justify-content: space-around;` —→ równe odległości między dziećmi
- `justify-content: space-evenly;` —→ wszystkie odległości między dziećmi i brzegami równe

\*pozostałe opcje nie są jeszcze wspierane przez wszystkie przeglądarki

Flex-grow - daje możliwość “rozpychania” się elementowi, jeżeli jest to możliwe

- Przyjmuje wartość numeryczną, relatywną, bez jednostki
- Wartości te służą jako proporcje elementów wypełniających dany “pojemnik”
- Jeżeli wszystkie elementy dostaną wartość “1”, wszystkie będą miały ten sam rozmiar
- Jeżeli tylko jeden dostanie jakąkolwiek wartość, np “1”, to będzie on próbował “rozepchać” się i pozostałe elementy skurczą się na ile to będzie możliwe.

Otwórz plik example3.html i sprawdź jak zachowują się elementy.  
Zmodyfikuj kod, by <main> było zawsze 2x wyższe niż header i footer.

```
.el-container {  
  display: flex;  
}  
  
.el1 {  
  flex-grow: 2;  
}  
  
.el2 {  
  flex-grow: 1  
}
```

## Zadanie 2

Otwórz plik Zad\_2/starter.html. Odwzoruj layout z plików w folderze Zad\_2/1.designs stosując wszystko, czego się dzisiaj nauczyłeś.

Nie musisz odwzorowywać bardzo dokładnie.

Pamiętaj, żeby sprawdzić czy na innych rozdzielczościach Twoja strona także wygląda dobrze!

Na górze ekranu każdego screena jest wartość w pikselach.

# CSS Grid

- Siatka strony, podzielona na komórki
- Używana do tworzenia layoutów całych stron
- Razem z Flexboxem tworzą zestaw wzajemnie uzupełniających się narzędzi do tworzenia responsywnych stron internetowych
- Używa jednostki *fr* (fraction), czyli “część wolnej przestrzeni”. Jest to jednostka relatywna. Koncept podobny co wartości numeryczne w flex-grow
- Można także używać zwykłych jednostek, które już znacie
- Domyślnie Grid próbuje umieścić każdy element w 1 komórce. Można zmienić te zachowanie

```
.grid-container {  
  display: grid;  
}
```

# CSS Grid - wybrane właściwości

## grid-template-columns

- Definiuje liczbę i rozmiary kolumn

- Liczba wpisanych wartości to liczba kolumn/wierszy
- Możemy używać różnych jednostek: px, fr, auto
- By nie wpisywać wartości dla każdej kolumny/wiersza, gdy nasz grid zawiera ich wiele, można zastosować funkcję repeat(liczba-elementów, rozmiar-elementów)

## grid-template-rows

- Definiuje liczbę i rozmiary wierszy

```
grid-template-columns: 1fr 2fr 1fr;  
grid-template-columns: 300px repeat(5, 1fr);  
grid-template-columns: auto 2fr auto;
```

# Zadanie 1

Otwórz plik Zad\_1/starter/starter.html i edytuj kod CSS by utworzyć grid:

- 3 kolumny, 3 wiersze
- Pierwsza kolumna musi mieć szerokość 350px, reszta wypełnić pozostałe im miejsce
- Wiersze wszystkie w takim samym rozmiarze, użyj funkcji repeat()
- Po wykonaniu zadania, włącz DevTools (f12) i zobacz jak grid reaguje na zmiany rozmiaru ekranu



## grid-auto-rows

- Używamy, gdy nie wiemy ile wierszy będzie nam potrzebne (na przykład elementy są dynamicznie generowane przez JavaScript)
- Możemy zdefiniować rozmiar i liczbę wierszy, których jesteśmy pewni poprzez *grid-template-rows*, a ewentualną resztę dodaną później przez *grid-auto-rows*

```
grid-template-rows: 300px;  
grid-auto-rows: 1fr;
```

1 wiersz o wysokości 300px

Reszta wierszy (wedle  
potrzeby) o wysokości 1fr  
każdy

# Inne wybrane właściwości

```
column-gap: 10px;  
row-gap: 10px;
```

Przerwy między kolumnami  
i wierszami  
(example3.html)

```
minmax(150px, auto);
```

Funkcja określająca minimalną i maksymalną  
szerokość kolumny, lub wysokość wiersza  
(example3.html)

```
grid-template-columns: 350px 1fr minmax(100px, auto);
```

Rozciąganie elementu  
(komórki) na siatce  
(example4..html)

```
.grid-container {  
  width: 100%;  
  height: 100vh;  
  border: 5px solid #909090;  
  display: grid;  
  grid-template-rows: 300px;  
  grid-auto-rows: 1fr;  
  grid-template-columns: 350px 1fr 1fr;  
  grid-template-areas:  
    "header header header"  
    "sidebar . ."  
    "sidebar . ."  
}  
  
.grid-child.header {  
  grid-area: header;  
}  
  
.grid-child.sidebar {  
  grid-area: sidebar;  
}
```

Zaznaczenie elementu  
(komórki) i wizualne  
przedstawienie  
rozciągnięcia(example4.html)

# Alternatywa do grid-template-areas

Prostsza do zrozumienia alternatywą są właściwości:

```
.grid-child.header {  
  grid-column-start: 1;  
  grid-column-end: 4;  
  /* grid-column-end: -1 */  
}  
  
.grid-child.sidebar {  
  grid-row-start: 2;  
  grid-row-end: 4;  
  /* grid-row-end: -1 */  
}
```

Początek od brzegu kolumny 1

Koniec na brzegu kolumny 4 (kolumny 1,2,3)

Wartość `-1` oznacza ostatnią kolumnę/wiersz

Dla row wartość `-1` nie działa na Chrome i Edge,  
mimo bycia w dokumentacji (grudzień 2022)

Reszta komórek wypełni pozostałe w  
gridzie miejsce, wedle zdefiniowanych  
kolumn i wierszy

example5.html

# Zadanie 2

Utwórz grid:

- 2x2
- z przerwami pomiędzy elementami 10px
- kolumny i wiersze równej szerokości/wysokości
- utwórz breakpoint dla ekranów szerokich na  $\leq 400\text{px}$ , gdzie grid utworzy 1 kolumnę i odpowiednią ilość wierszy (podpowiedź: Media Queries)

## \*Zadanie 3

Utwórz grid:

- 5 kolumn (pierwsza 2x większa od reszty)
- Automatyczna ilość wierszy, równej wysokości, ale minimum 50px (pomoc: źródło 11.e, "Sizing Functions")
- Pierwsza komórka (klasa header) rozciągnięta na całą szerokość strony jako header
- Druga komórka rozciągnięta od 2 wiersza, zabierająca łącznie 10 wierszy jako sidebar po lewej stronie

header				
sidebar				

# CSS Multicol

- Multiple column layout daje możliwość utworzenia kolumn na stronie o wyglądzie jak w gazetach
- Przydatny w szczególnych przypadkach, nie ma przewagi nad Flexboxem i Gridem
- W większość przypadków lepiej używać Flexboxa i Grida

```
.newspaper-text {  
  column-count: 3;  
  column-gap: 20px;  
  column-rule-style: solid;  
  column-rule-width: 3px;  
  column-rule-color: red;  
}
```

→  
Lorem ipsum dolor sit amet, consectetur  
adipiscing elit, sed diam nonummy nibh  
euismod tincidunt ut laoreet dolore magna  
aliquam erat volutpat. Ut wisi enim ad minim  
veniam, quis nostrud exerci tation  
ullamcorper suscipit lobortis nisl ut aliquip ex

ea commodo consequat. Duis autem vel eum  
iriure dolor in hendrerit in vulputate velit esse  
molestie consequat, vel illum dolore eu  
feugiat nulla facilisis at vero eros et accumsan  
et iusto odio dignissim qui blandit praesent  
luptatum zzril delenit augue duis dolore te

feugait nulla facilisi. Nam liber tempor cum  
soluta nobis eleifend option congue nihil  
imperdiet doming id quod mazim placerat  
facer possim assum.

# Bootstrap

- Framework CSS
- W praktyce jest to paczka gotowego kodu CSS, który aplikujemy poprzez nadawanie odpowiednich klas elementom
- Przyspiesza tworzenie strony
- Prosty w obsłudze, wszystkie zawiłości dzieją się “pod maską”
- Możliwe jest nadpisywanie stylów aplikowanych przez Bootstrap, swoim własnym CSSem i dodawanie swoich własnych
- Czasem łatwo rozpoznać strony robione przy użyciu Bootstrapa, gdy deweloper nie użyje swoich stylów, tylko pozostawi domyślne
- Może spowalniać stronę ponieważ przeglądarka musi ściągnąć cały kod Bootstrapa, nawet jeśli wszystkiego nie wykorzystuje

# Instalacja

- Jest kilka możliwości instalacji Bootstrapa, najprostszy (i najmniej wydajny) to dodanie go do strony poprzez CDN (Content Delivery Network) (źródło 13.d)
- O innych metodach przeczytasz w źródle 13.a
- Pliki z ćwiczeniami w tym rozdziale mają już dodanego Bootstrapa przez CDN

**Include Bootstrap's CSS and JS.** Place the `<link>` tag in the `<head>` for our CSS, and the `<script>` tag for our JavaScript bundle (including Popper for positioning dropdowns, poppers, and tooltips) before the closing `</body>`. Learn more about our [CDN links](#).

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" re
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.
  </body>
</html>
```



- Bootstrap stworzony został z myślą o podejściu mobile-first i RWD

100% szerokości ekranu

Ciemne tło

Biały tekst

Padding i wartość

```
<div class="container-fluid bg-dark text-white p-5">Lorem ipsum dolor sit amet consectetur, adipisicing elit. Porro explicabo cum iste eum rem quod vel repellendus odit architecto eius!</div>  
<div class="container bg-dark text-white p-5">Lorem ipsum dolor, sit amet consectetur adipisicing elit. Labore error saepe reiciendis. Quia eligendi aliquid dolor possimus fugiat architecto expedita.</div>
```

Szerokość “na sztywno”, ale zmieniająca wartość, zależnie od szerokości ekranu

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	X-Large ≥1200px	XX-Large ≥1400px
.container	100%	540px	720px	960px	1140px	1320px

	<b>Extra small</b> <576px	<b>Small</b> ≥576px	<b>Medium</b> ≥768px	<b>Large</b> ≥992px	<b>X-Large</b> ≥1200px	<b>XX-Large</b> ≥1400px
<code>.container</code>	100%	540px	720px	960px	1140px	1320px
<code>.container-sm</code>	100%	540px	720px	960px	1140px	1320px
<code>.container-md</code>	100%	100%	720px	960px	1140px	1320px
<code>.container-lg</code>	100%	100%	100%	960px	1140px	1320px
<code>.container-xl</code>	100%	100%	100%	100%	1140px	1320px
<code>.container-xxl</code>	100%	100%	100%	100%	100%	1320px
<code>.container-fluid</code>	100%	100%	100%	100%	100%	100%

Np. będzie miał 100% szerokości do 992px, a potem przejdzie na szerokość “na sztywno”

Dokumentacja Bootstrapa (źródło 13.a) posiada wyszukiwarke, dzięki której łatwo można wyszukać interesujące nas style.

Do tego jest on wiekowym już narzędziem, więc odpowiedzi na wiele pytań znajdziecie w Google.

Pamiętajcie, że na zajęciach używamy Bootstrapa w wersji 5.x, więc poradniki sprzed kilku lat mogą być nieaktualne.

# Zadanie 1

- Dzielimy się na grupy 3 osobowe
- Cała grupa przeszukuje dokumentację, a jedna osoba edytuje kod, by je rozwiązać
- Po rozwiązaniu zadania, edytujący dzieli się kodem z szukającymi

Utwórz stronę w Bootstrapie w pliku Zad\_1/starter/starter.html opierając się na tym designie:



mobile



PC

# Tailwind

- Kolejny framework CSS
- O wiele młodszy, ale zyskujący na popularności
- Oferuje więcej opcji stylowania bez konieczności nadpisywania jego stylu swoim kodem CSS
- Trudniejszy w nauce od Bootstrapa
- Może sprawiać problemy początkującym deweloperom, którzy nie znają dobrze CSS
- Łatwa integracja z popularnymi obecnie frameworkami JavaScript
- Można go dodać do strony poprzez CDN, jednak sprawdzi się to tylko w małych projektach, używających podstawowych funkcji Tailwinda

# Pomocy! Tyle do nauczenia! Czy ja to wszystko muszę znać?

Nie musisz znać wszystkiego! By wykonać aplikacje na zaliczenie studiów sprawdź wymagania co do technologii. Absolutnie konieczna jest wiedza z zakresu zajęć, punkty 1-11. Punkty 12-14 są “opcjonalne”. Sprawdź wymagania projektu zaliczeniowego, czy musisz używać frameworka CSS. Pamiętaj, że nie musisz wkuwać wszystkiego na pamięć. Nikt tego nie robi. Nawet doświadczeni deweloperzy czytają dokumentacje i szukają odpowiedzi w Google!

Co umieć z dzisiejszych zajęć by znaleźć pracę? Jak wyżej, punkty 1-10 + dobrze jest znać podstawy któregoś frameworka CSS. Przejrzyj oferty dla juniorów na stronach typu justjoin.it i zobacz jakie technologie są wymagane.

# Zadania bonusowe

Zadania ogólne, bez szczegółowych wymagań. Musisz utworzyć sam/sama wszystkie potrzebne pliki i wykorzystać wiedzę, źródła, prezentację i wyszukiwarke Google do rozwiązania problemu.

# Zadanie bonusowe 1

Utwórz tabelę 4x3, wycentruj ją na stronie. Dostosuj tabelę tak, by wyglądała dobrze na każdej szerokości ekranu.



## Zadanie bonusowe 2

Skopiuj/edytuj pliki poprzedniego zadania i dodaj do strony header oraz footer. Header ma być “przyklejony” cały czas w tym samym miejscu ekranu. Dostosuj stronę by nadal była responsywna.

## Zadanie bonusowe 3

Stwórz animację za pomocą @keyframes.

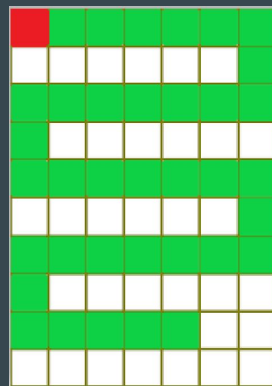
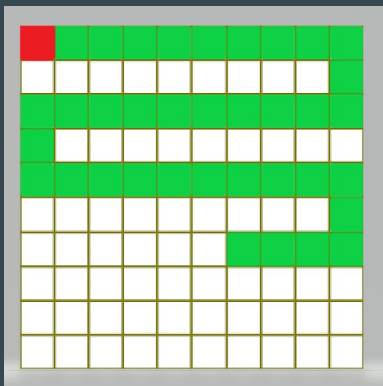
Na ekranie o szerokości >500px element ma poruszać się po stronie w prawo i w lewo.

Na mniejszym ekranie ma poruszać się w górę i w dół

## Zadanie Bonusowe 4 \*

Podziel cały viewport na kwadraty za pomocą dowolnej techniki (np CSS Grid) o szerokości minimum 10 kwadratów o stałej wielkości.

Pokoloruj poszczególne komórki, by układały się w wijącego się od jednej strony ekranu do drugiej węża. Wąż powinien mieścić się na każdej szerokości ekranu, jednocześnie wijąc się jak najbardziej na boki. Pamiętaj, żeby ilość kwadratów, którą posiada wąż się nie zmieniła - przecież się nagle nie skurczy / nie urośnie ;)



# Wykonałaś/eś zadanie bonusowe nr 4?

Zrób deploy strony na np [Netlify](#) lub [Github Pages](#)

Napisz do mnie maila z linkiem do repozytorium na GitHubie na [bednarekdaniel8@gmail.com](mailto:bednarekdaniel8@gmail.com).

Chętnie sprawdzę i dam znać co myślę o Twoim rozwiązaniu!

# Projekty

1.

Utwórz responsywną stronę ze zdjęciami. Jako źródła obrazów możesz użyć strony lorem ipsum <https://picsum.photos/>

Umieść na niej co najmniej 20 zdjęć i header rozciągnięty na całą szerokość strony

Nie używaj frameworków

2.

Utwórz responsywną stronę z 4 artykułami (każdy ma posiadać zdjęcie, tytuł i tekst długi na ok 300 słów)

Dodaj header rozciągnięty na całą szerokość strony oraz footer na dole strony

Użyj Bootstrapa

3.

\* Utwórz responsywną stronę z 4 zdjęciami, każde z tytułem.

Dodaj header i footer rozciągnięte na całą szerokość strony. Footer musi być na dole strony

Nie używaj frameworków lub użyj Tailwinda (na własną rękę, czytając dokumentację)