Ćwiczenie 1 – „Pierwsza siatka: 3 kolumny, autouzupełnianie wierszy i odstępy”.

Celem jest zrozumienie absolutnych podstaw: czym jest *grid container*, *grid items*, jak ustawić kolumny, przerwy (*gap*) i zachowanie domyślnego auto-układania elementów.

Na czym oparte jest ćwiczenie?

- display: grid zmienia element rodzica w *kontener siatki*, a jego bezpośrednie dzieci automatycznie stają się *elementami siatki* (grid items). [w3schools.com], [w3schools.com]
- Grid to **dwuwymiarowy** system (wiersze i kolumny); w najprostszym wariancie deklarujemy liczbę/rozmiary kolumn przez grid-template-columns, wiersze tworzą się w razie potrzeby automatycznie. [w3schools.com]
- Warto używać gap do ustawiania odstępów między wierszami i kolumnami. [w3schools.com]

Rezultat po ćwiczeniu

- Zbudujesz prosty układ z nagłówkiem i siatką kart (3 kolumny na szerszych ekranach).
- Zrozumiesz różnicę między kontenerem i elementem siatki.
- Nauczysz się zmieniać liczbę kolumn i odstępy.

Dlaczego to działa?

- .grid { display: grid; } — zamienia .grid w *grid container*; wszystkie bezpośrednie .card stają się *grid items* (nie trzeba nic dopisywać na dzieciach). [w3schools.com]
- grid-template-columns: auto auto auto; — jawnie definiuje trzy kolumny; gdy elementów jest więcej niż mieści się w jednym wierszu, siatka **automatycznie tworzy nowe wiersze** i układa elementy w porządku źródłowym (auto-placement). [w3schools.com], [w3schools.com]
- gap — ustawia jednostajne odstępy między torami siatki (wiersze i kolumny). [w3schools.com]

Krok 1 — Podmień typy jednostek kolumn

Zamiast auto, użyjemy elastycznych jednostek fr, które dzielą **pozostałą** przestrzeń kontenera.

```
.grid {
 display: grid;
 grid-template-columns: 1fr 1fr 1fr; /* 3 równe kolumny dzielące wolne miejsce */
 gap: 12px;
 padding: 12px;
}
```

- fr (fraction) to podstawowy sposób na *responsywne* kolumny w Gridzie. (W3Schools opisuje definiowanie szerokości kolumn poprzez wartości rozmiarów; fr należy do często używanych). [w3schools.com]

Mini-zadanie A
Zmień na grid-template-columns: 2fr 1fr 1fr; i zaobserwuj, która kolumna jest szersza.

Uwaga: **suma „ułamków”** decyduje o podziale wolnej szerokości.

Krok 2 — Zmieniaj liczbę kolumn

Spróbuj szybko „przeskalować” układ:

```
/* 2 kolumny */
.grid { grid-template-columns: 1fr 1fr; }

/* 4 kolumny */
.grid { grid-template-columns: 1fr 1fr 1fr 1fr; }
```

Cel dydaktyczny: Zauważ, że *kolumny definiujemy na kontenerze*, a elementy siatki automatycznie wypełniają kolejne „komórki”. [w3schools.com]

Krok 3 — Eksperyment z gap

Zmień odstępy, porównując gap: 0;, gap: 12px;, row-gap: 24px; column-gap: 8px;.

- gap to skrót dla row-gap i column-gap. [w3schools.com]

Mini-zadanie B
Ustaw row-gap: 24px i column-gap: 8px. Zwróć uwagę na wrażenie optycznego. Wskaż, który z odstępów wpływa na pion/poziom.

Krok 4 — Rola elementów siatki (opcjonalnie)

Dodajmy jednemu kafelkowi rozpiętość na **dwie kolumny** i **dwa wiersze**, by zasygnalizować możliwości po stronie *grid items*:

```
/* tymczasowo dodaj do <div class="card card--big">...</div> */
.card--big {
  grid-column: 1 / span 2; /* od linii kolumny 1, na 2 kolumny */
  grid-row: 1 / span 2;    /* od linii wiersza 1, na 2 wiersze */
}
```

- Własności grid-column i grid-row pozwalają określać, **od której linii** zacząć i **ile kolumn/wierszy** „zająć” (skrót dla par start/end; można używać słowa kluczowego span). [w3schools.com]

Mini-zadanie C
Dodaj klasę .card--big do kafelka „1” i zobacz, jak przesuwają się pozostałe elementy przez auto-układanie.

Krok 5 — Krótka wersja „responsywna” z minmax() (preview kolejnych ćwiczeń)

Podmień definicję kolumn na:

```
.grid {
 display: grid;
 grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
 gap: 12px;
}
```

- To sprawia, że liczba kolumn **dostosowuje się** do szerokości ekranu (gdy mniej miejsca, spada do 2 lub 1 kolumny, ale karta nigdy nie jest węższa niż 180px). (Mechanika grid-template-columns oraz torów/kolumn – nawiązanie do podstaw z W3Schools; repeat()/minmax() rozwiniemy w następnym ćwiczeniu). [w3schools.com]

Krok 6 — Dodaj *progressive enhancement* z @supports (fallback bez grida)

Dla starszych przeglądarek można zadeklarować prosty układ „blokowy”, a **lepszy** włączyć, jeśli przeglądarka wspiera display: grid:

```
/* Fallback bez wsparcia dla Grid – elementy pod sobą */
.grid {
 display: block;
}
.grid .card {
 margin: 12px;    /* proste odstępy jako zamiennik gap */
}

/* Jeśli przeglądarka obsługuje grid – użyjemy pełnego układu */
@supports (display: grid) {
 .grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 12px;
 }
}
```

- @supports pozwala stosować reguły warunkowo – tylko gdy dana właściwość/wartość jest wspierana (tu: display: grid). Zostawiamy prosty styl **poza** blokiem @supports jako bezpieczny fallback. [w3schools.com]

Pytania kontrolne (do dyskusji z klasą)

1. Co sprawia, że .grid staje się *grid container* i co to oznacza dla jego dzieci? (oczekiwana odpowiedź: display: grid, dzieci stają się *grid items*). [w3schools.com]
2. Jak przeglądarka zachowuje się, gdy elementów jest więcej niż mieści jeden wiersz? (tworzy nowe wiersze i **auto-rozmieszcza** elementy). [w3schools.com]
3. Czym różni się gap od marginesów elementów? (działa między torami siatki; nie „wypycha" obwiedni kontenera). [w3schools.com]
4. Jak grid-column i grid-row wpływają na pozycjonowanie pojedynczego elementu? (pozwalają mu rozciągać się na określoną liczbę linii/komórek). [w3schools.com]
5. Do czego służy @supports i gdzie powinny być style „zapasowe"? (warunkowe stosowanie nowocześniejszych reguł; fallback **poza** blokiem @supports). [w3schools.com]

Zadanie do realizacji samodzielnie (krótka modyfikacja)

- Zmień grid-template-columns tak, by środkowa kolumna była **dwukrotnie** szersza niż pozostałe.
- Ustaw różne row-gap i column-gap i opisz efekt.
- Dodaj jednemu kafelkowi rozpiętość grid-column: 2 / span 2; i zaobserwuj, jak zmienia się układ. [w3schools.com]

Kod początkowy strony:

```
<!doctype html>
<html lang="pl">
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1" />
 <title>Grid – Ćwiczenie 1</title>
 <style>
  /* Reset pudełkowania – nieobowiązkowe, ale ułatwia pracę z szerokościami */
  * { box-sizing: border-box; }

  body {
   margin: 0;
   font-family: system-ui, -apple-system, Segoe UI, Roboto, Arial, sans-serif;
   line-height: 1.5;
  }

  header {
   padding: 1rem;
   background: #0b69c7;
   color: #fff;
   text-align: center;
  }

  /* 1) Uczyń .grid kontenerem siatki */
  .grid {
   display: grid;              /* kluczowa deklaracja – tworzy grid container */
   grid-template-columns: auto auto auto; /* 3 kolumny o automatycznej szerokości */
   gap: 12px;                  /* odstęp między wierszami i kolumnami */
   padding: 12px;
   max-width: 1000px;
   margin-inline: auto;
  }

  /* Elementy siatki – każde bezpośrednie dziecko .grid */
  .card {
   background: #f6f8fa;
   border: 1px solid #e0e6ee;
   border-radius: 8px;
   padding: 16px;
   text-align: center;
   font-weight: 600;
  }
```

```
 </style>
</head>
<body>
 <header>
  <h1>Moja pierwsza siatka CSS Grid</h1>
  <p>3 kolumny + auto-układanie wierszy</p>
 </header>

 <section class="grid">
  <div class="card">1</div>
  <div class="card">2</div>
  <div class="card">3</div>
  <div class="card">4</div>
  <div class="card">5</div>
  <div class="card">6</div>
  <div class="card">7</div>
  <div class="card">8</div>
  <div class="card">9</div>
 </section>
</body>
</html>
```