Etap 1: Budujemy szkielet w HTML (plik `index.html`)

Krok 1: Podstawa dokumentu Najpierw musimy powiedzieć przeglądarce, w jakim języku piszemy, i przygotować ramy dokumentu.

HTML
```
<!DOCTYPE html>
<html lang="pl">
```

- `<!DOCTYPE html>` – To informacja dla przeglądarki: „używamy najnowszego standardu HTML5”.
- `<html lang="pl">` – Otwieramy główny znacznik strony i od razu mówimy, że strona jest po polsku (dzięki temu np. tłumacz Google nie będzie wciskał na siłę swoich okienek).

Krok 2: Głowa strony (Head) Teraz dodajemy część niewidoczną dla użytkownika, ale kluczową dla działania strony.

HTML
```
<head>
    <meta charset="UTF-8">
    <title>Moja pierwsza strona</title>
    <link rel="stylesheet" href="style.css">
</head>
```

- `<meta charset="UTF-8">` – Dzięki temu polskie znaki (ą, ę, ł) będą wyświetlać się poprawnie, zamiast "krzaczków".
- `<title>` – Tekst, który pojawi się na karcie przeglądarki.
- `<link...>` – Kluczowy moment: tutaj łączymy nasz plik HTML z plikiem CSS, który za chwilę stworzymy.

Krok 3: Ciało strony (Body) i Główny Kontener Otwieramy część wizualną. Wszystko, co tu wpiszemy, będzie widać na ekranie.

HTML
```
<body>
    <div id="container">
```

- `<body>` – Otwiera widoczną część strony.
- `<div id="container">` – Tworzymy niewidzialne „pudełko”, do którego wrzucimy całą resztę. Dlaczego? Bo dużo łatwiej jest potem wycentrować jedno duże pudełko na środku ekranu, niż każdy element z osobna.

Krok 4: Nagłówek i Menu Budujemy górę naszej strony.

HTML

```
<header>
    <h1>Tytuł strony</h1>
</header>

<nav>
    Tutaj będzie menu nawigacyjne
</nav>
```

- Używamy znaczników semantycznych (<header>, <nav>). Działają dokładnie tak samo jak , ale mówią robotom Google'a i czytnikom ekranu: „tu jest nagłówek”, „tu są linki”.

Krok 5: Środek z dwiema kolumnami

HTML

```
<main>
    <aside id="col-left">
        Lewa kolumna
    </aside>

    <section id="col-center">
        Prawa kolumna z treścią
    </section>
</main>
```

- <main> – Opakowanie na naszą główną treść. Wrzucamy do niego dwa elementy: lewą kolumnę (nazwijmy ją <aside>, co oznacza panel boczny) i środek (<section>).

Krok 6: Stopka i zamknięcie całości Na koniec zamykamy wszystkie tagi, które otworzyliśmy.

HTML

```
<footer>
    Moja stopka
</footer>

    </div> </body>
</html>
```

Etap 2: Układamy klocki i stylujemy(nadajemy kolory) w CSS (plik `style.css`)

Strona w HTML wygląda na razie jak zwykły tekst. Przechodzimy do pliku CSS, żeby to poukładać.

Krok 1: Reset przeglądarki Przeglądarki domyślnie dodają własne marginesy. Zawsze na początku warto je wyzerować, żeby mieć pełną kontrolę.

CSS
```
body {
    margin: 0;
    background-color: #333333; /* Ustawiamy ciemne tło poza kontenerem */
    color: #ffffff; /* Biały tekst domyślnie */
}
```

Krok 2: Centrowanie kontenera Teraz bierzemy to nasze wielkie „pudełko" (w którym jest cała strona) i ustawiamy na środku.

CSS
```
#container {
    max-width: 1078px;
    margin: 0 auto;
    background-color: #000000; /* Czarne tło samej strony */
}
```

- `max-width: 1078px;` – Zamiast sztywnego `width`, dajemy szerokość maksymalną. Dzięki temu na małych ekranach strona się zwęzi, a nie utnie.
- `margin: 0 auto;` – To jest zapis w CSS, który odpowiada za centrowanie. Oznacza: zero marginesu od góry/dołu i *automatyczny* margines po bokach (co fizycznie spycha pudełko na sam środek ekranu).

Krok 3: Kolory i odstępy podstawowych elementów Nadajmy nagłówkowi, menu i stopce trochę przestrzeni i koloru, żeby były widoczne.

CSS
```
header, nav, footer {
    padding: 20px; /* Robimy odstęp wewnątrz, żeby tekst nie przyklejał się do
krawędzi */
    text-align: center; /* Centrujemy tekst wewnątrz tych bloków */
    border-bottom: 1px solid #444; /* Dajemy cienką linię na dole dla
oddzielenia */
}

#col-left, #col-center {
    padding: 20px;
}
```

Krok 4: KLUCZOWY MOMENT – Układanie kolumn (Flexbox) To tutaj zastępujemy starsze rozwiązanie (floaty) nowszym (Flexbox).

Jak położyć lewą i prawą kolumnę obok siebie? Wystarczy chwycić ich rodzica (czyli `<main>`) i włączyć mu tryb elastyczny (Flexbox).

CSS
```
main {
    display: flex;
}
```

- I już! Samo wpisanie `display: flex;` sprawia, że wszystkie dzieci elementu `<main>` (czyli nasze kolumny) stają w jednym rzędzie, obok siebie. Żadnego psucia stopki, żadnego "clearowania".

Krok 5: Szerokość kolumn Kolumny stoją obok siebie, ale musimy im jeszcze powiedzieć, ile miejsca mają zająć.

CSS
```
#col-left {
    flex: 0 0 200px;
    background-color: #006acc;
}
```

- `flex: 0 0 200px;` – To nowoczesny sposób zapisu. Oznacza: nie rośnij, nie kurcz się, miej dokładnie **200 pikseli** szerokości.

Zapis `flex: 0 0 200px;` to tak zwana właściwość skrócona (shorthand) w systemie Flexbox. Pozwala ona zapisać trzy różne komendy w jednej krótkiej linijce.

Oto dokładne rozwinięcie tego zapisu, czytając od lewej do prawej:

1. Pierwsze zero (`flex-grow: 0`) – ZAKAZ ROZCIĄGANIA

Ta wartość odpowiada za apetyt na wolne miejsce. Zapisanie zera oznacza: **"Jeśli w kontenerze zostanie pusta przestrzeń, ten element ma po nią nie sięgać"**. Nawet jeśli obok jest mnóstwo miejsca, kolumna nie urośnie ani o piksel.

2. Drugie zero (`flex-shrink: 0`) – ZAKAZ KURCZENIA

Ta wartość określa zachowanie w sytuacji kryzysowej. Zapisanie zera oznacza: **"Jeśli w kontenerze zabraknie miejsca (np. użytkownik ogląda stronę na wąskim telefonie), ten element nie może się skurczyć"**. Będzie on twardo trzymał swój rozmiar jak skała, nawet jeśli reszta strony będzie musiała się ścisnąć.

3. Ostatnia wartość (`flex-basis: 200px`) – ROZMIAR BAZOWY

To jest punkt wyjścia. Mówi przeglądarce: **"Idealna, domyślna szerokość tego elementu to 200 pikseli"**. Od tej wartości Flexbox zaczyna obliczać, co zrobić z elementem, zanim zastosuje zasady rośnięcia (grow) lub kurczenia (shrink).

Podsumowując w jednym zdaniu: Ten zapis to rozkaz dla przeglądarki: **"Masz mieć dokładnie 200 pikseli szerokości. Nie wolno ci urosnąć ani się skurczyć, bez względu na to, co dzieje się na ekranie"**.

Jest to idealne i najczęściej stosowane ustawienie dla bocznych menu, pasków nawigacji czy stałych banerów reklamowych.

CSS

```
#col-center {
    flex: 1;
    background-color: #ffffff;
    color: #000000; /* Zmieniamy tekst na czarny, bo tło jest białe */
}
```

- `flex: 1;` – To kolejny bardzo ważny zapis. Oznacza: „Weź całą pozostałą przestrzeń, jaka została w kontenerze”. Jeśli kontener ma 1078px, a lewa kolumna wzięła 200px, to środek automatycznie zajmie resztę, bez żadnego ręcznego liczenia!

1. Kiedy masz tylko jednego chętnego na wolne miejsce (NASZ przypadek)

W naszym kodzie mieliśmy taką sytuację:

- **Lewa kolumna (`#col-left`)** dostała sztywny zakaz rozciągania się: `flex: 0 0 200px;`.
- **Środkowa kolumna (`#col-center`)** dostała pozwolenie na rozciąganie: `flex: 1;`.

Zostało np. 800 pikseli wolnego miejsca. Przeglądarka pyta: *"Kto chce to wolne miejsce?"*. Zgłasza się tylko środkowa kolumna. Ponieważ nie ma żadnej konkurencji, bierze 100% wolnego miejsca. Nie ma znaczenia, czy powiesz przeglądarce "daj środkowej kolumnie 1 udział", czy "daj jej 10 udziałów" – i tak zgarnia całą pulę, bo jest sama. Dlatego `1` i `10` zadziałało tak samo. Wpisuje się `1` dla czystości i prostoty kodu.

2. Kiedy zaczyna się walka o miejsce (Kiedy 1 vs 10 ma znaczenie)

Wartości te nabierają sensu, gdy masz **więcej niż jeden element**, który chce się rozciągać. Wyobraź sobie, że dzielicie pizzę (wolne miejsce na ekranie) i ustalacie proporcje:

- Jeśli Kolumna A ma `flex: 1`, a Kolumna B ma `flex: 1`:
  - Suma udziałów to 2. Kolumna A dostaje 1/2 pizzy, Kolumna B dostaje 1/2.
- Jeśli Kolumna A ma `flex: 1`, a Kolumna B ma `flex: 3`:
  - Suma udziałów to 4. Kolumna A dostaje 1/4 pizzy, Kolumna B dostaje 3/4.