ЛАБОРАТОРНА РОБОТА №7
Робота з Flexbox в CSS

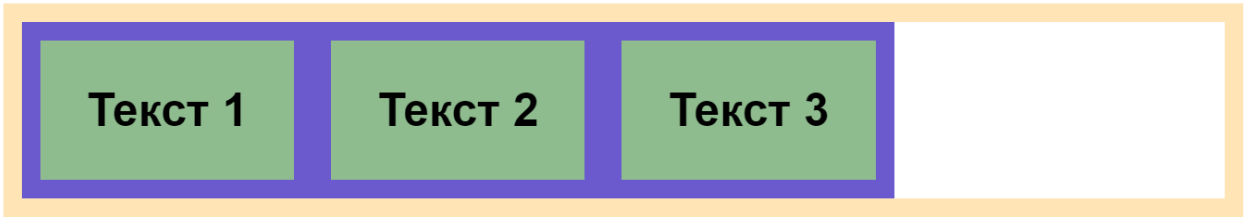
Теоретичний матеріал

Перелік основних CSS-властивостей flex-контейнера:

- **display: flex;**

Визначає блоковий flex-контейнер, та перетворює всі свої прямі та дочірні елементи у flex-елементи.

Після застосування даного параметру flex-елементи вишиковуються в рядок, а flex-контейнер залишається блочним (тобто займає всю ширину простору в яку він поміщений, у нас це вікно браузера).



При зміні ширини екрана, flex-елементи не будуть «ломатись», тобто переміщатись на інший рядок, вони «ніби склеїлись» між собою.

- **display: inline-flex;**

Визначає рядковий flex-контейнер, та перетворює всі свої прямі та дочірні елементи у flex-елементи.

Після застосування даного параметру flex-елементи вишиковуються в рядок, а flex-контейнер стає рядковим (тобто перестає займає всю ширину простору в яку він поміщений, у нас це вікно браузера) і переймає всі значення властивості display: inline-block;

Якщо елементу-батька всього контейнера задати вирівнювання тексту по центру, то весь flex-контейнер також вирівняється по центру разом із своїми flex-елементами, що є неможливим при display: flex;

```
.block {
    text-align: center;
}
.block__row {
    display: inline-flex;
```

- **justify-content**

Повернемось до display: flex; і як ми бачимо, за замовчуванням, наші flex-елементи вишиковуються в рядок.

Властивість justify-content відповідає за положення flex-елемента в середині flex-контейнера та визначає вирівнювання flex-елементів вздовж основної осі (напрямок залежить від значення властивості flex-direction), проте, за замовчуванням, це вісь «x» (вирівнювання по горизонталі).

Значення:

– justify-content: flex-start;

За замовчуванням justify-content містить значення flex-start, яке прижимає flex-елементи до початку осі, тому його можна не вказувати, оскільки всі flex-елементи розташовуються на початку flex-контейнера, тобто з лівої сторони.

```
.block__row {
    display: flex;
    justify-content: flex-start;
```

– justify-content: flex-end;

Для тощо, щоби розташувати всі flex-елементи справа (в кінці flex-контейнера), потрібно задати значення flex-end.

```
.block__row {
    display: flex;
    justify-content: flex-end;
```

– justify-content: center;

Для тощо, щоби розташувати всі flex-елементи по центру flex-контейнера, потрібно задати значення center.

```
.block__row {
    display: flex;
    justify-content: center;
```

– justify-content: space-between;

Flex-елементи розміщуються поступово по основній осі: перший елемент знаходиться на початку осі, останній елемент знаходиться наприкінці осі, та між всіма flex-елементами встановлюється однакова відстань (простір) між ними.

```
.block__row {
    display: flex;
    justify-content: space-between;
```

Проте непотрібно плутати із внутрішніми відступами (padding) між блоками. Якщо ми зменшимо розмір екрану, то ми побачимо, що ніяких відступів між блоками немає, зменшується або збільшується саме простір між блоками.

– justify-content: space-around;

Flex-елементи розташовуються рівномірно по основній осі. До кожного flex-елементу додається простір ліворуч і праворуч.

```
.block__row {
    display: flex;
    justify-content: space-around;
```

Ми бачимо, що для кожного flex-елемента додається однаковий простір як зліва так і справа.

Відповідно між flex-елементами 1 та 2, а також між 2 та 3, буде по два простори.

Також непотрібно плутати із внутрішніми відступами (padding) між блоками.

Цей простір між flex-елементами з'являється тільки тоді, коли це дозволяє розмір flex-контейнера.

– justify-content: space-evenly;

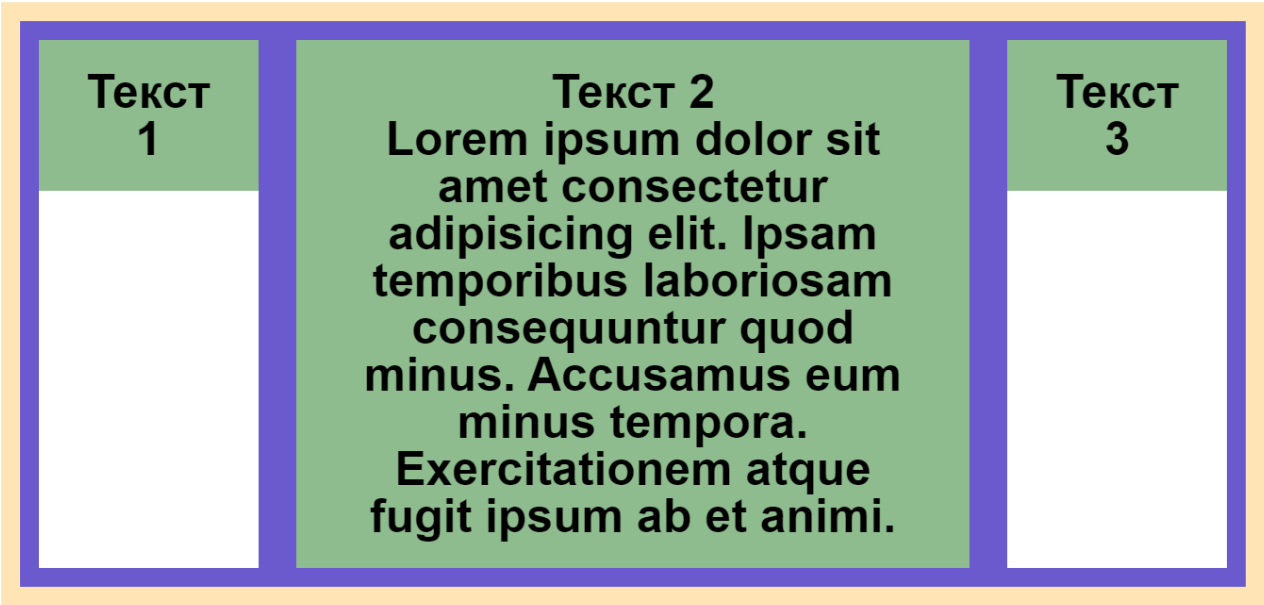
Flex-елементи розташовуються рівномірно по основній осі з однаковим простором навколо них.

```
.block__row {
    display: flex;
    justify-content: space-evenly;
```

- **align-items**

Визначає вирівнювання flex-елементів вздовж поперечної осі (напрямок залежить від значення властивості flex-direction), проте, за замовчуванням, це вісь «y» (вирівнювання по вертикалі).

Приклад: є три блоки з різним текстовим наповненням.



– align-items: stretch;

За замовчуванням align-items містить значення stretch, flex-елементи розтягуються на весь розмір поперечної осі flex-контейнера, тому його можна не вказувати.

```
.block__row {
    display: flex;
    align-items: stretch;
```

З попереднього прикладу ми бачимо, що хоча контент 1-го та 3-го flex-елементів менші за контент 2-го flex-елемента, самі flex-елементи будуть підлаштовуватись під висоту самого високого flex-елемента. Тобто, немає значення, чи контент самого високого flex-елемента містить за умовою більше контенту чи йому задано іншу висоту.

– align-items: flex-start;
Flex-елементи притискаються до початку поперечної осі.

```
.block__row {
    display: flex;
    align-items: flex-start;
```

– align-items: flex-end;
Flex-елементи притискаються до кінця поперечної осі.

```
.block__row {
    display: flex;
    align-items: flex-end;
```

– align-items: center;
Flex-елементи розташовуються в центрі поперечної осі, тобто всі flex-елементи розташуються по горизонтальному центру відносно самого високого flex-елемента.

```
.block__row {
    display: flex;
    align-items: center;
```

– align-items: baseline;
Розташовує по базовій лінії всі flex-елементи, проте перед тим необхідно збільшити кількість контенту 2-го flex-елемента.

```
.block__row {
    display: flex;
    align-items: baseline;
```

- **flex-wrap**
Визначає, чи зможуть flex-елементи переміщатися на наступні рядки, коли перестають поміщатися всередині flex-контейнера.

Значення:
– flex-wrap: nowrap;
Значення за замовчуванням (не потрібно вказувати). Flex-елементи не можуть переміщатися на наступні рядки flex-контейнера, коли їм бракує місця.

```
.block__row {
    display: flex;
    flex-wrap: nowrap;
```

– flex-wrap: wrap;
Flex-елементи можуть переміщатися на наступні рядки flex-контейнера.

```
.block__row {
    display: flex;
    flex-wrap: wrap;
```

– flex-wrap: wrap-reverse;

Flex-елементи зможуть переміщатися на наступні рядки flex-контейнера, але у зворотному порядку.

```
.block__row {
    display: flex;
    flex-wrap: wrap-reverse;
```

- **flex-direction**

Встановлює основну вісь, таким чином визначає напрямок розташування flex-елементів у flex-контейнері.

Значення:

– flex-direction: row;

Значення за замовчуванням (не потрібно вказувати). Основна вісь горизонтальна. Всі flex-елементи розташуються ліворуч по горизонталі або в рядок.

```
.block__row {
    display: flex;
    flex-direction: row;
```

– flex-direction: row;

Всі flex-елементи розташуються вкінці flex-контейнера, у зворотному порядку по горизонталі.

```
.block__row {
    display: flex;
    flex-direction: row-reverse;
```

– flex-direction: column;

Основна вісь зміниться з горизонтальної на вертикальну, всі flex-елементи розташуються зверху донизу в колонку.

```
.block__row {
    display: flex;
    flex-direction: column;
```

Питання: в чому відмінність між конструкціями display:flex, flex-direction:column та звичайними тегами div без флексів?

Якщо забрати display:flex та flex-direction:column, то нічого не зміниться. У нас будуть звичайні блочні теги div, які займають всю ширину вікна батька.

Поєднання display:flex з flex-direction:column дозволяє нам робити із flex-елементами ті самі маніпуляції, які ми виконуємо при значенні flex-direction:row.

А тепер більш детально: те що ми бачимо на екрані, а саме поведінку наших flex-елементів, то це поведінка за замовчуванням, яка встановлена для всього flex-контейнера через властивість align-items:stretch. Саме це заставляє наші flex-елементи займати всю ширину flex-контейнера.

Тому, якщо ми встановлено для flex-контейнера властивість align-items:flex-end, то ми побачимо, що розташування flex-елементів зміниться, всі вони притиснуться до правої межі flex-контейнера, хоча раніше всі flex-елементи притискались до нижньої межі flex-контейнера.

```css
.block__row {
    display: flex;
    flex-direction: column;
    align-items: flex-end;

    border: 20px solid #FFE4B5;
    margin: 0px 0px 20px 0px;
}
```

Справа в тому, що властивість flex-direction:column, крім того, що змінює розташування flex-елементів з горизонтального на вертикальне, розташовуючи їх в колонку, вона також змінює ролі у властивостей align-items та justify-content. Тому, якщо раніше align-items керував поведінкою flex-елементів по вертикалі, а justify-content керував поведінкою flex-елементів по горизонталі, то flex-direction:column заставляє властивості align-items та justify-content «ніби помінятись» ролями. Хоча насправді, нічого не змінюється, просто при задані властивості flex-direction:column ми фактично «перевернули» наш flex-контейнер. Але всі інші властивості працюють без змін, тобто властивість justify-content керує поведінкою flex-елементів по основній осі, якою тепер стає вісь вертикалі, а властивість align-items керує поведінкою flex-елементів по побічній осі, якою тепер стає вісь горизонталі.

– flex-direction: column-reverse;

Основна вісь зміниться з горизонтальної на вертикальну, та всі flex-елементи розташуються знизу догори.

```css
.block__row {
    display: flex;
    flex-direction: column-reverse;
```

Перелік основних CSS-властивостей flex-елементів:

- **align-self**

Подібна до align-items, проте застосовується тільки до flex-елемента.

Отже, що робить властивість align-self? Ми вже знаємо, що значення за замовчуванням властивості align-items: stretch, тобто flex-елементи розтягуються на весь розмір поперечної осі flex-контейнера, іншими словами, самі flex-елементи будуть підлаштовуватись під висоту самого високого flex-елемента.

Властивість align-self дозволяє перевизначити (переписати) вказане значення властивості align-items, яка вказується для всіх flex-елементів, що містяться у flex-контейнері, для конкретного flex-елемента, задаючи його індивідуальне вирівнювання вздовж поперечної осі (напрямок залежить від значення властивості flex-direction).

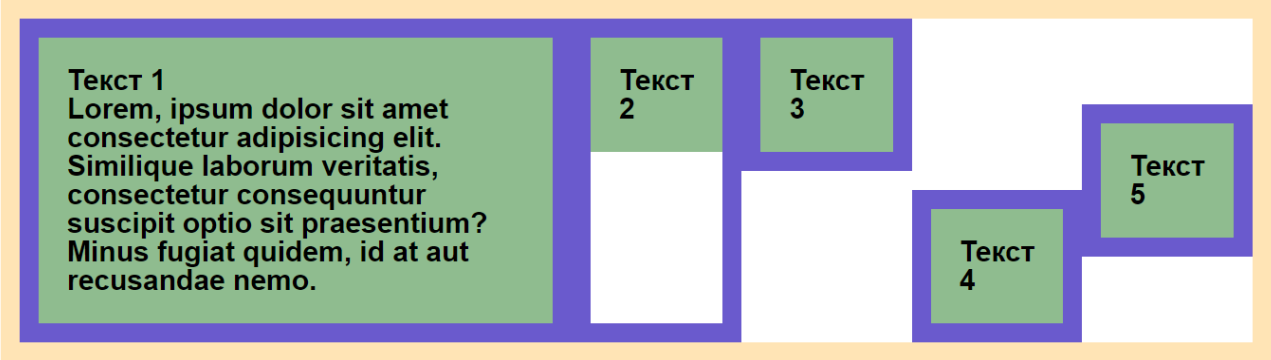
Властивість align-self має ті ж самі значення, що й властивість align-items.

– stretch – значення за замовчування (не потрібно вказувати), flex-елементи розтягуються на весь розмір поперечної осі flex-контейнера.
– flex-start – flex-елементи притискаються до початку поперечної осі.
– flex-end – flex-елементи притискаються до кінця поперечної осі.
– center – flex-елементи розташовуються в центрі поперечної осі.
– auto – flex-елементи приймають місце розташування flex-контейнера, тобто значення властивості align-items.

Приклад:

```
.block__element1{
}
.block__element2{
    align-self: stretch;
}
.block__element3{
    align-self: flex-start;
}
.block__element4{
    align-self: flex-end;
}
.block__element5{
    align-self: auto;
}
```

Результат.



- **order**

Керує порядком, в якому розташовуються flex-елементи всередині flex-контейнера. Це можуть бути: прямі напрямки, зворотні напрямки, чи перемішка порядку слідування flex-елементів.

Навіть, якщо flex-елемент не змінює свій порядок виводу, ми все рівно задаємо йому значення order з його порядковим номером. Бо якщо order не задавати, то тоді flex-елементи, які не містять параметр order, автоматично зсуваються на початок поперечної осі.

Приклад. Вивести flex-елементи згідно такого порядку: 3-2-1-5-4

```
.block__element1{
    order: 3;
}
.block__element2{
    align-self: stretch;
    order: 2;
}
.block__element3{
    align-self: flex-start;
    order: 1;
}
.block__element4{
    align-self: flex-end;
    order: 5;
}
.block__element5{
    align-self: auto;
    order: 4;
}
```

- **flex-basis**

Визначає розмір flex-елемента за замовчуванням, до початку розподілу простору, що залишився.

Значення (можуть бути у відсотках, пікселях тощо).

– flex-basis: auto;

Значення за замовчування (не потрібно вказувати). Flex-елемент займатиме розмір залежно від контенту всередині нього.

– flex-basis: значення в пікселях;

Встановлюємо flex-елементам початкову ширину в пікселях.

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 50px;
}
```

– flex-basis: значення у відсотках;

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 30%;
}
```

Зауважте, що властивість flex-basis не є аналогом властивості width, а є базовим значенням розміру.

Ми можемо також, задавати для кожного flex-елемента його розмір.

```
.block__column {
    border: 20px solid #6A5ACD;
}
.block__element1{
    flex-basis: 500px;
}
.block__element2{
}
.block__element3{
}
.block__element4{
    flex-basis: 250px;
}
.block__element5{
}
```

Або у відсотках.

```
.block__column {
    border: 20px solid #6A5ACD;
}
.block__element1{
    flex-basis: 40%;
}
.block__element2{
}
.block__element3{
}
.block__element4{
    flex-basis: 20%;
}
.block__element5{
}
```

- **flex-grow**

Визначає можливість flex-елемента збільшуватись у розмірі, відповідно до його базового розміру (flex-basis), при необхідності заповнюючи весь flex-контейнер: 0 – заборонено, 1 – дозволено.

Значення:

– flex-grow: 0;

Значення за замовчування (не потрібно вказувати). Flex-елементу заборонено збільшуватись за значення flex-basis.

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 500px;
    flex-grow: 0;
}
```

– flex-grow: 1;

Flex-елемент збільшиться, заповнюючи весь простір flex-контейнера.

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 500px;
    flex-grow: 1;
}
```

Ми можемо також, задавати для кожного flex-елемента можливість його збільшення.

В даному прикладі, flex-елементи 1, 3 та 5 будуть мати статичний розмір 500px, а flex-елементи 2 та 4 збільшать свої розміри за рахунок вільного простору flex-контейнера, і таким чином flex-контейнер буде повністю покритий flex-елементами.

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 500px;
}
.block__element1{
}
.block__element2{
    flex-grow: 1;
}
.block__element3{
}
.block__element4{
    flex-grow: 1;
}
.block__element5{
}
```

- **flex-shrink**

Визначає можливість flex-елемента зменшуватись у розмірі, відповідно до його базового розміру (flex-basis), при необхідності заповнюючи весь flex-контейнер: 0 – заборонено, 1 – дозволено.

Значення:

– flex-shrink: 1;

Значення за замовчуванням (не потрібно вказувати). Flex-елемент може зменшуватись за значення flex-basis.

Змінимо розміри браузера і ми побачимо, що flex-елементи зменшують свій розмір.

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 500px;
    flex-shrink: 1;
}
```

– flex-shrink: 0;

Flex-елемент не може зменшуватись за значення flex-basis.

```
.block__column {
    border: 20px solid #6A5ACD;
    flex-basis: 500px;
    flex-shrink: 0;
}
```

Змінимо розміри браузера і ми побачимо, що flex-елементи не зменшують свій розмір менше ніж значення flex-basis.

- **flex**

Властивість flex – це скорочений запис для flex-grow, flex-shrink та flex-basis.

Визначає можливість flex-елемента зменшуватися та збільшуватися у розмірі, за необхідності заповнюючи весь flex-контейнер (0 – заборонено, 1 – дозволено), а також задає базовий розмір flex-елемента.

Синтаксис:

flex: значення_flex-grow значення_flex-shrink значення_flex-basis;

```
.block__column {
    border: 20px solid #6A5ACD;
    flex: 0 1 auto;
}
```

ПРАКТИЧНА ЧАСТИНА

Завдання 1

Відтворити приклад макета додатків погоди на n-кількість днів, на базі гнучкої FLEX розмітки (використавши всі іконки). Приклад компонування наведено на рисунку «Завдання 7_1»

Завдання 2

Верстка веб-сторінки з фіксованими розмірами згідно макету (рисунок «Завдання 7_2») на базі гнучкої FLEX розмітки, використовуючи довільні зображення та довільний контент.