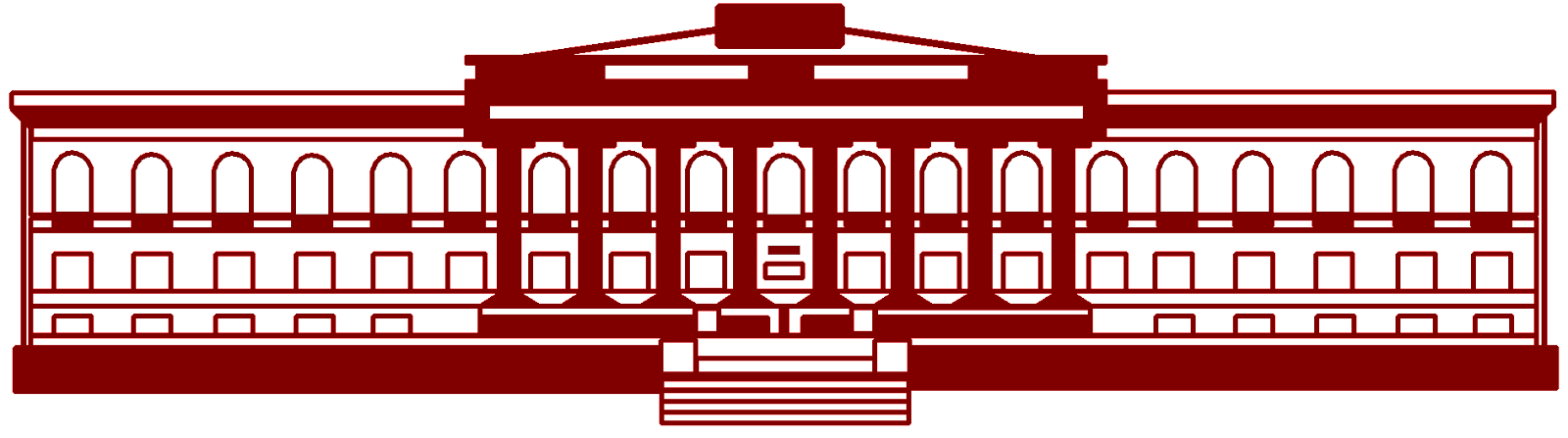
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій

**Кафедра прикладних інформаційних систем**



**Звіт**

до виконання лабораторної роботи №2

з дисципліни «**Веб-технології і програмування сайтів**»

на тему:

« Верстка сторінки та застосування стилів за допомогою CSS »

**Виконано:**

студ. групи ПП- 31, підгрупа 2

Шкандюк Анною Леонідівною

**Перевірено:**

Пирог М.В.

**Київ – 2023**

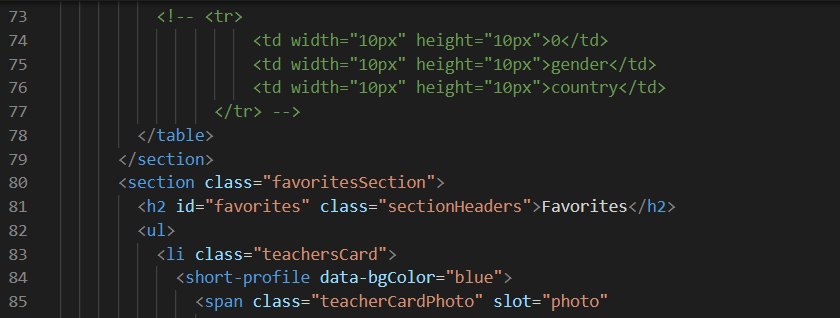
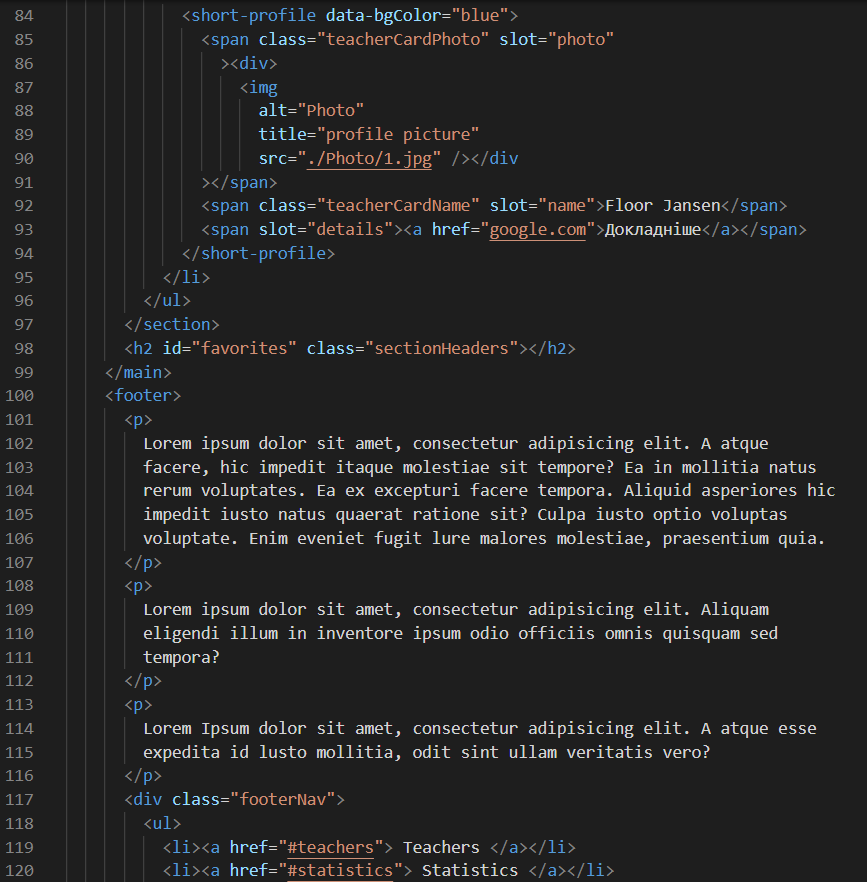
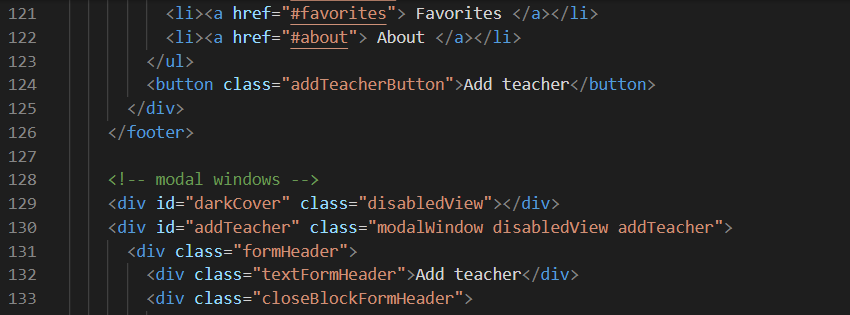
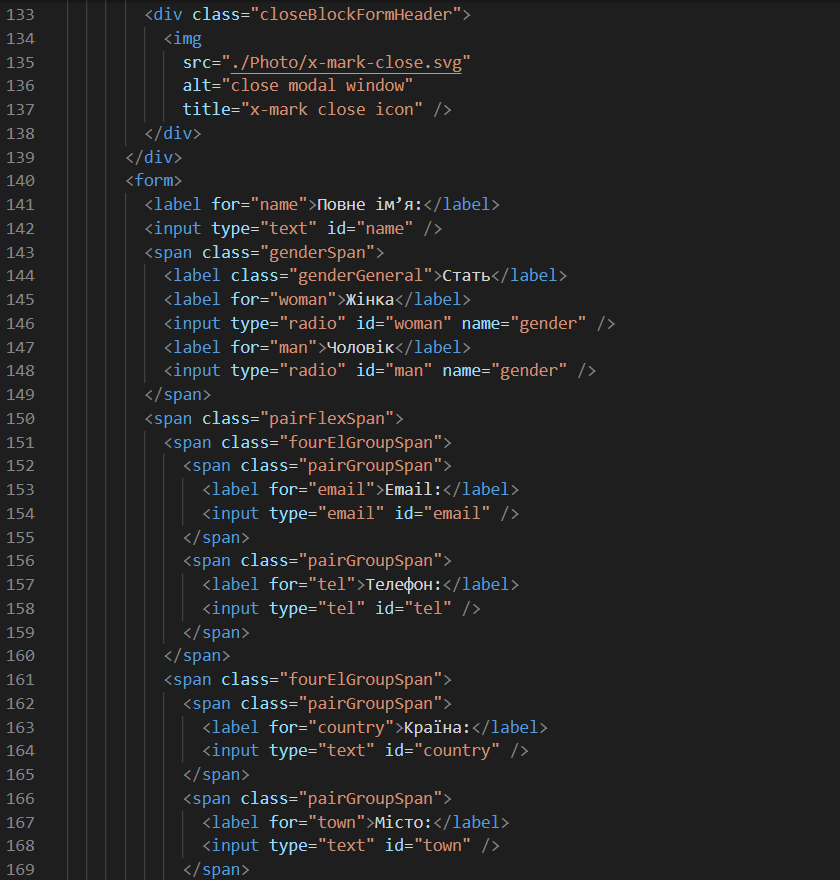
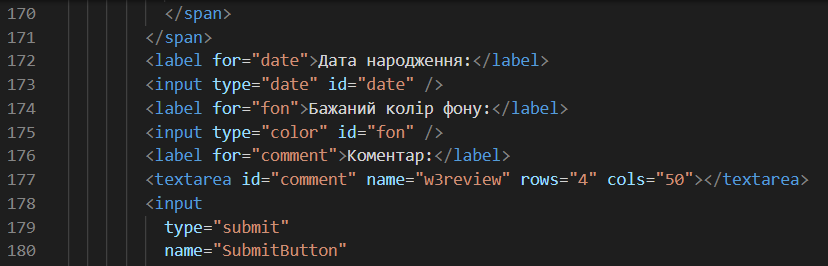
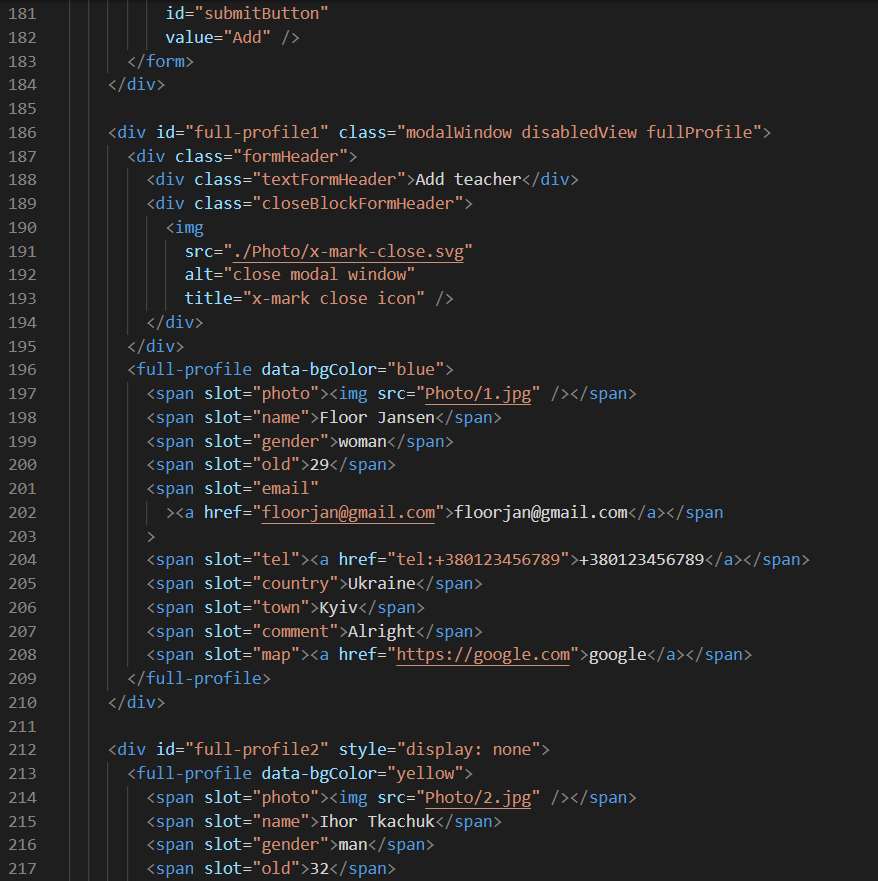
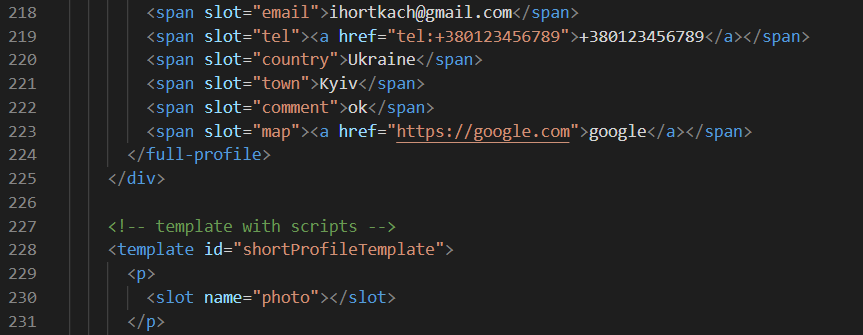
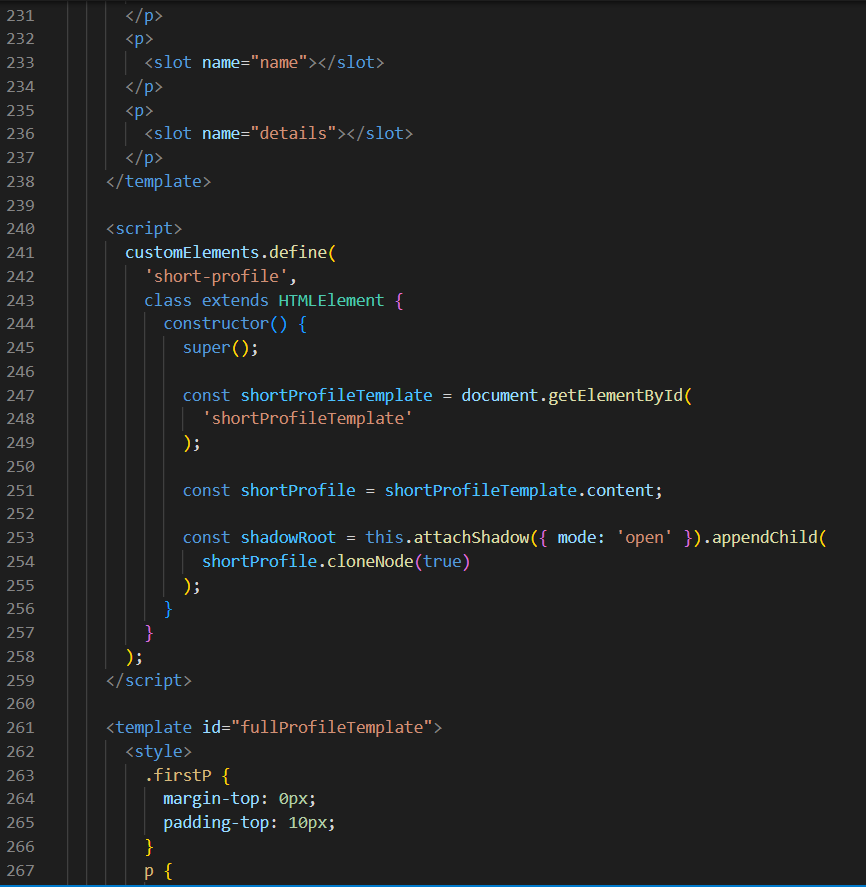
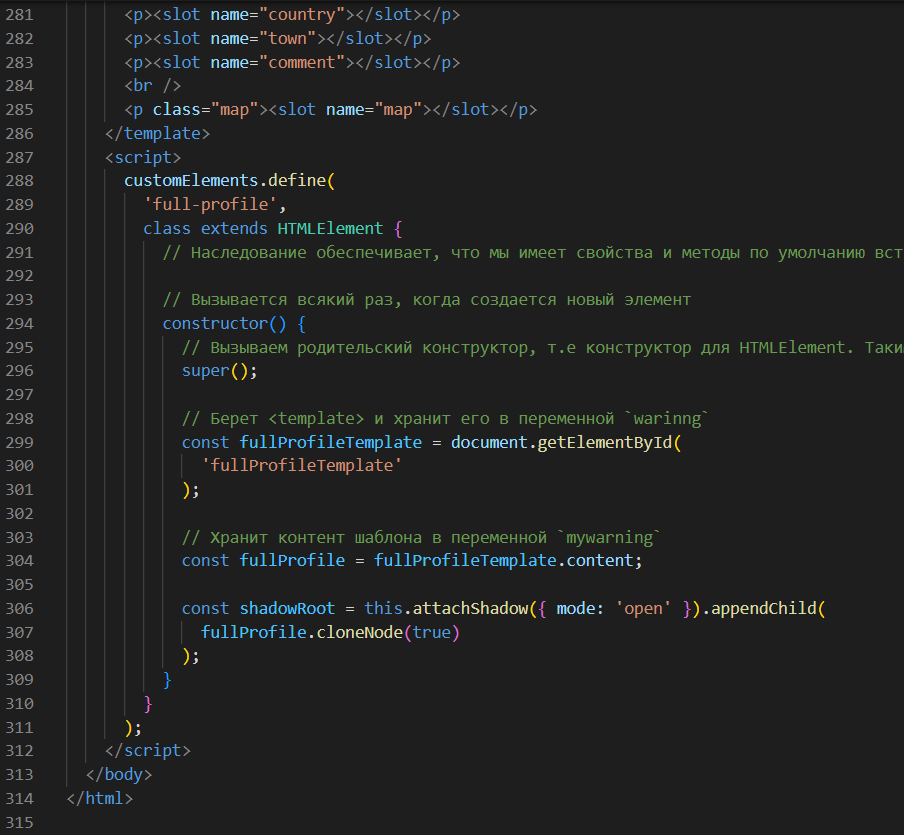
1. **Мета роботи:**

Отримати навички створення макету HTML-сторінки з використанням різних технік, познайомитися з технологією CSS.

1. **Завдання та їх виконання:**

**Завдання 1**. З окремих файлів, що були створені у ході виконання ЛР №1, створити один HTML-файл. Структурно цей файл має відповідати макету, що виступає як зразок результату, що необхідно отримати.



**Завдання 2**. Додати стилі до сторінки, використовуючи CSS (бажано використовувати препроцесор SCSS). Отриманий результат має бути ***дуже схожим*** на макет-зразок.

2.1. Список того, що має співпадати з макетом:

- всі загальні розміри сторінки (загальна ширина контенту, відступи до границь вікна браузера)

- всі використані шрифти, та їх розміри у кожному окремому випадку;

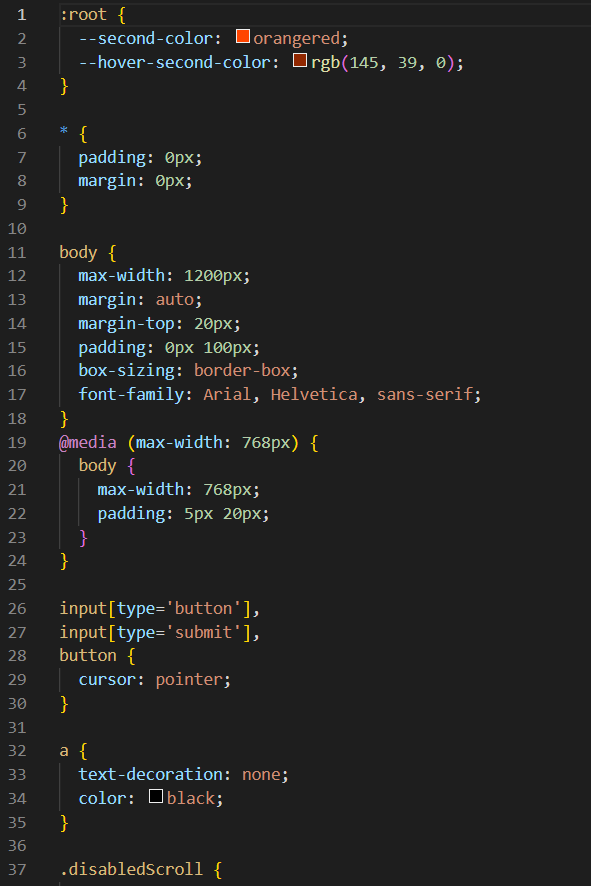
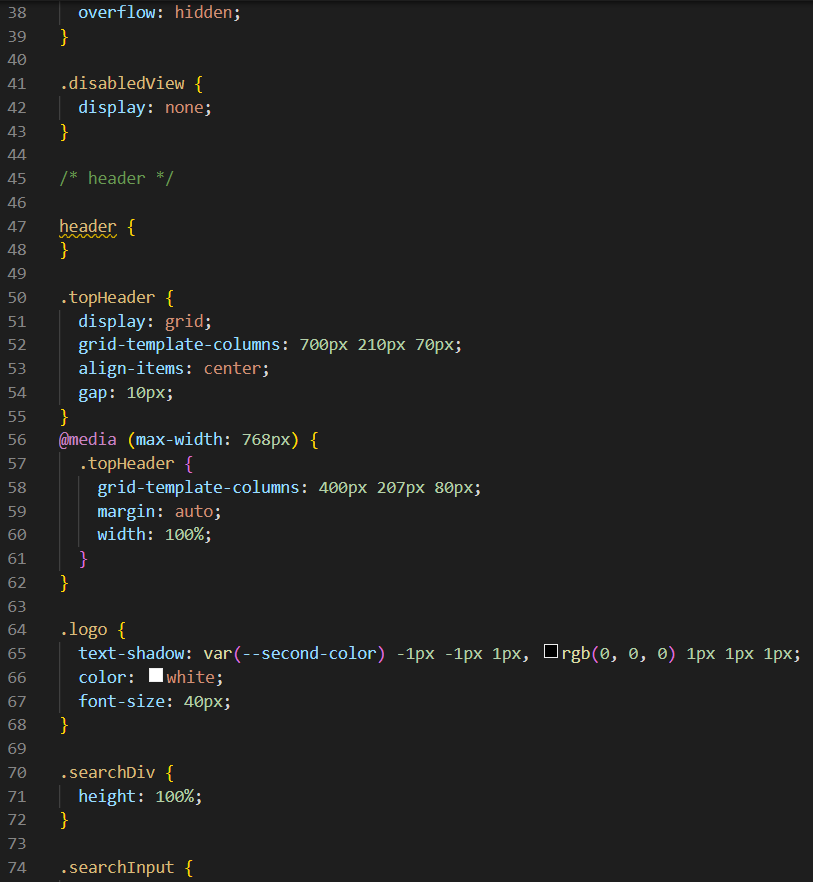
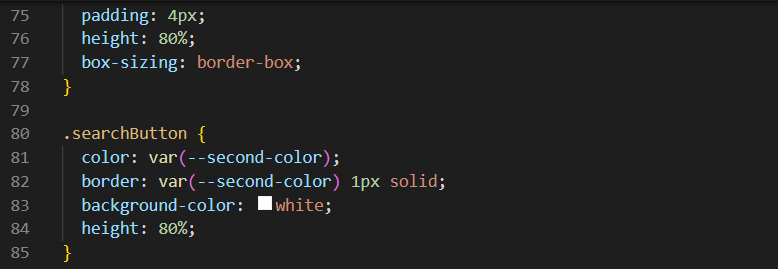
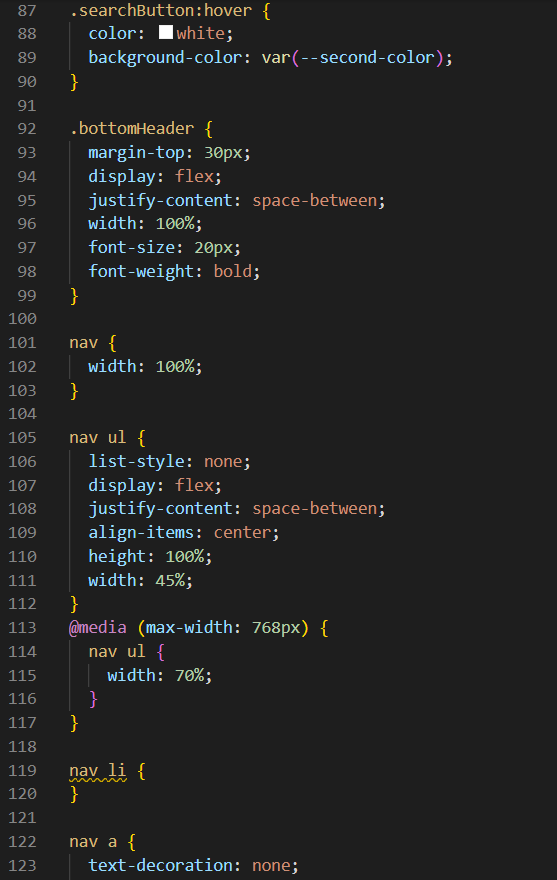
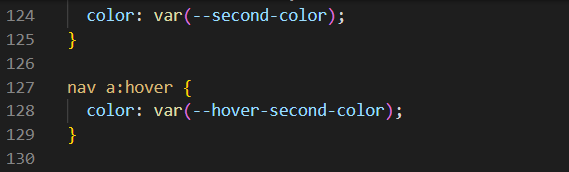
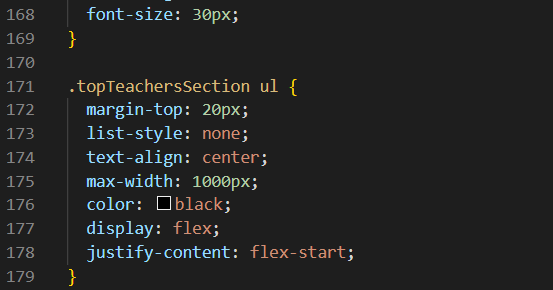
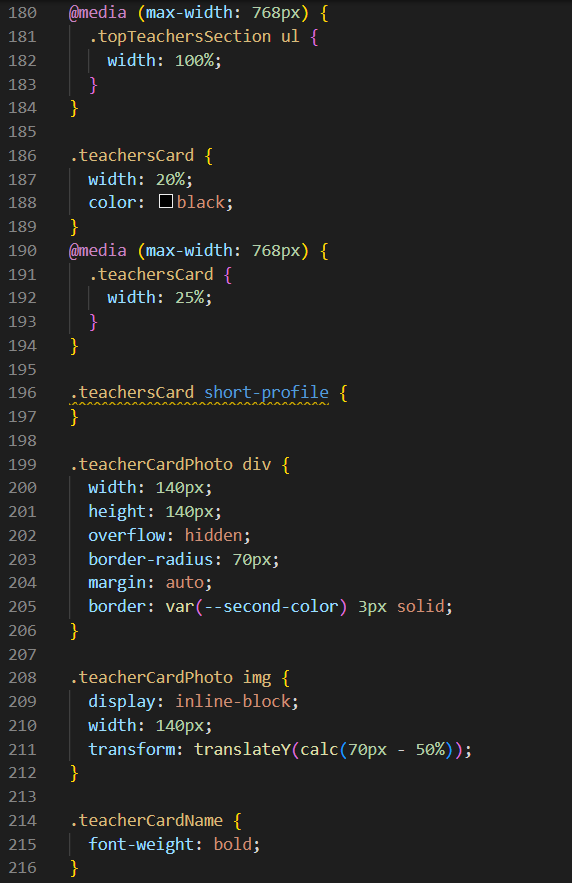
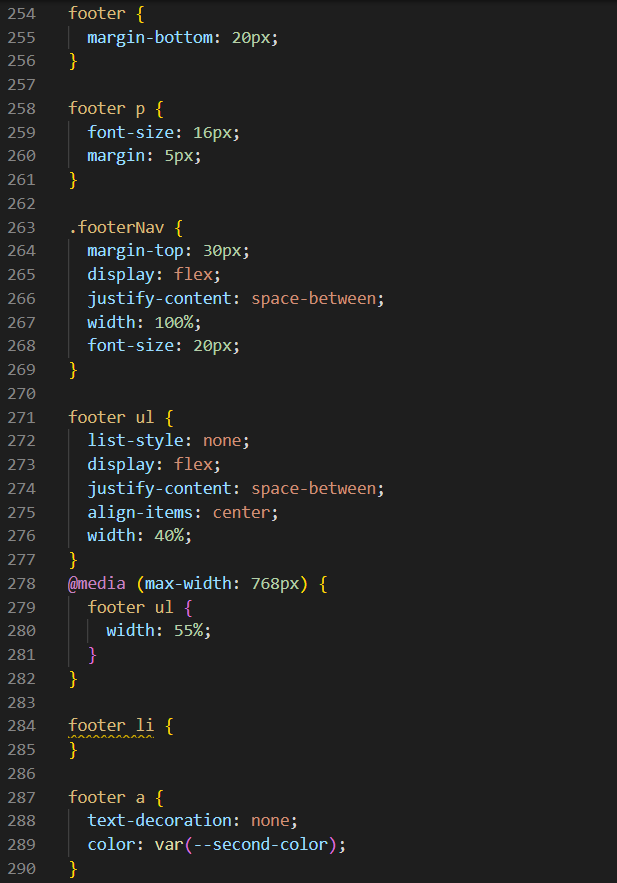
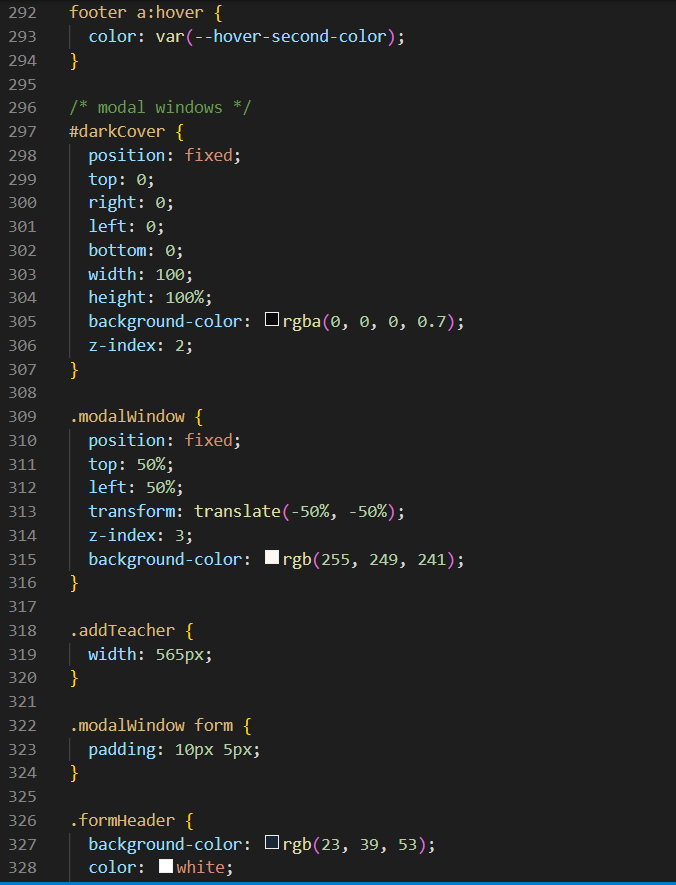
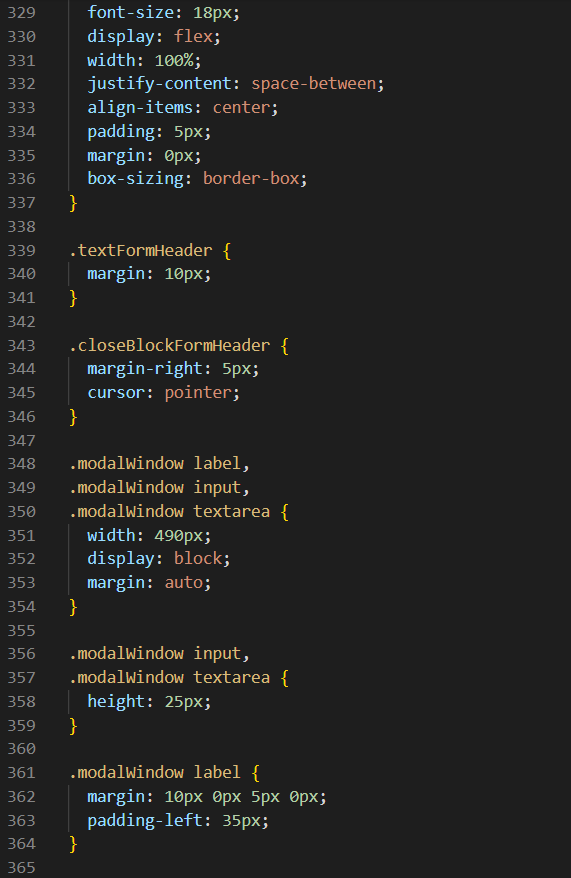
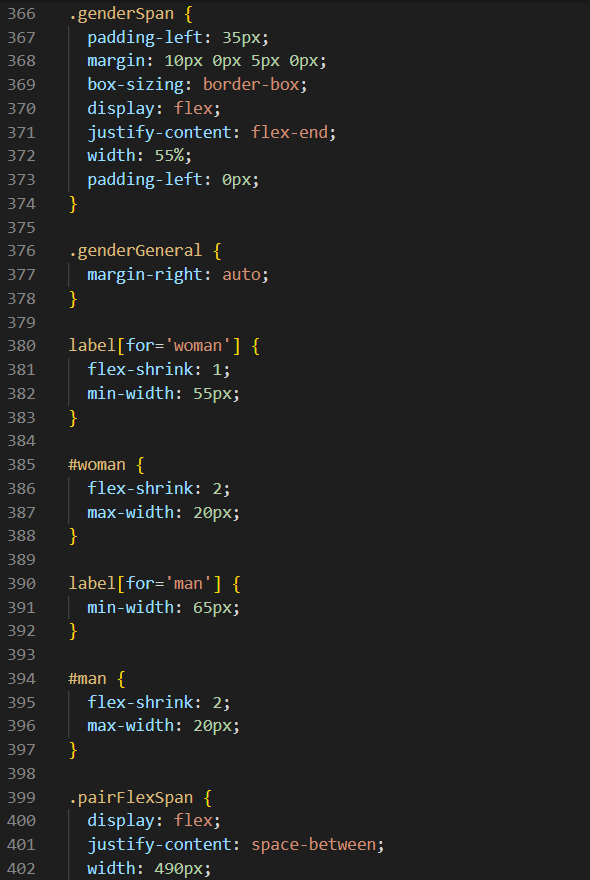
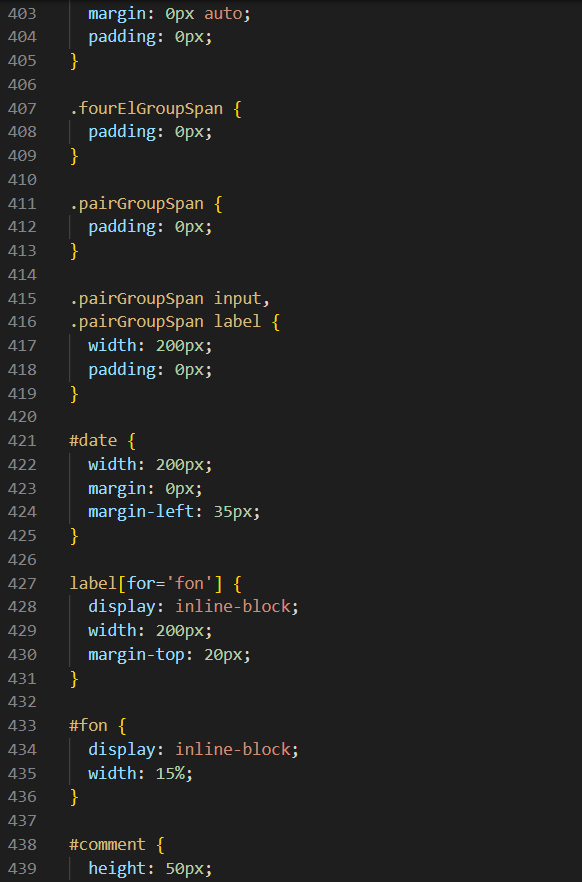
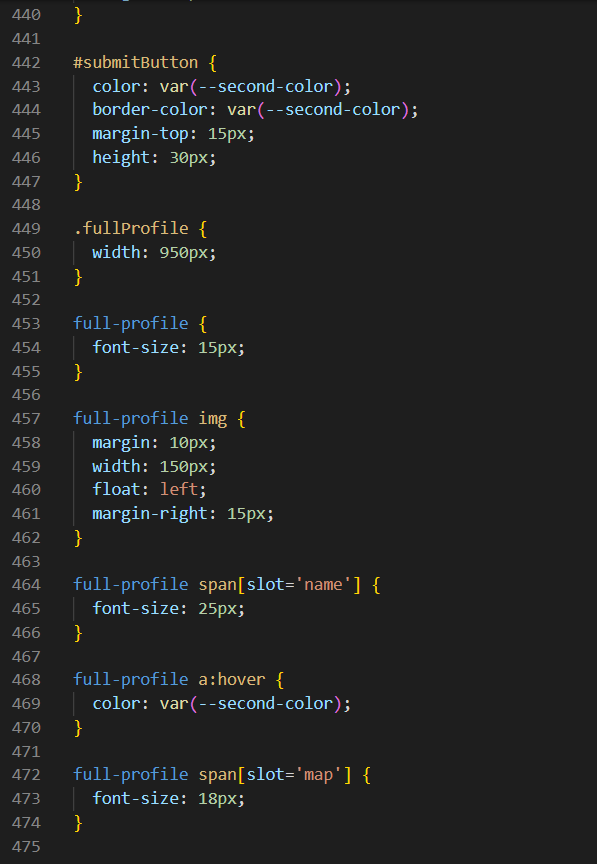
- всі кольори, в тому числі при наведенні миші;

- всі розміри зображень, елементів форм;

- всі відступи між будь-якими елементами (відступи між секціями, відступи між заголовком та текстом, відступи між елементами меню, тощо), та всередині елементів (наприклад, у кнопках).

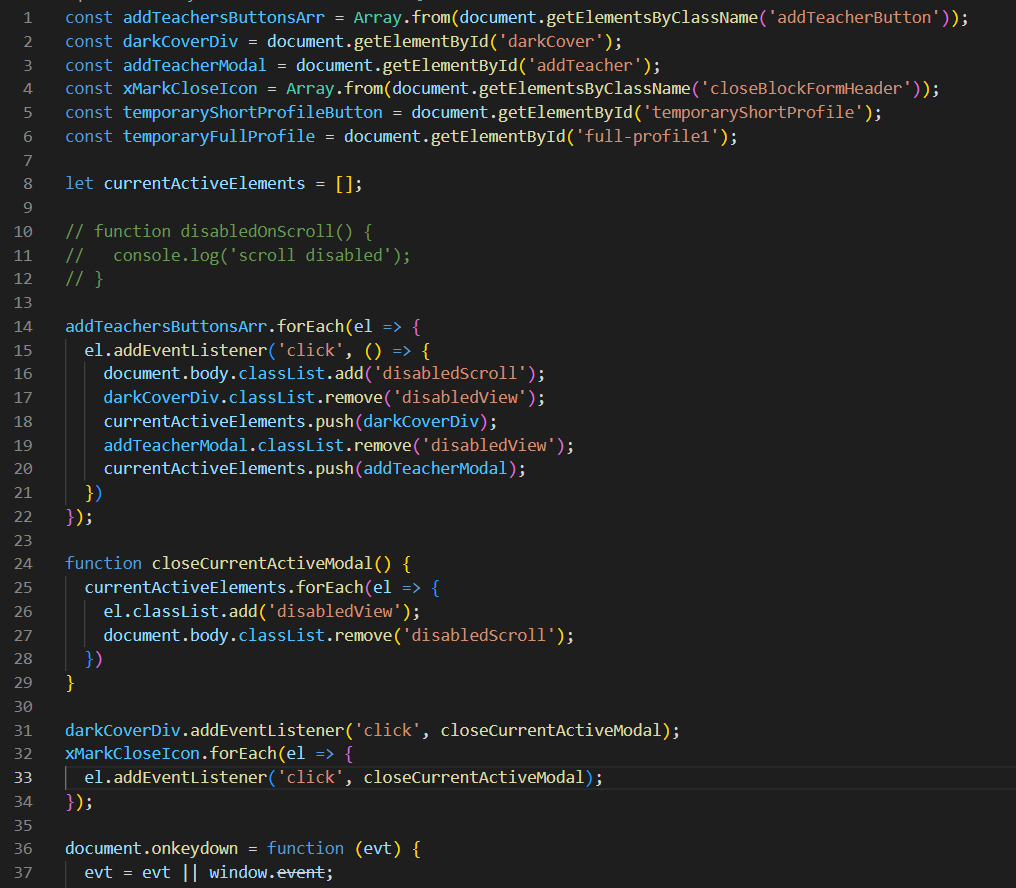
2.2. Для створення каркасу сторінки та побудови структури елементів необхідно використовувати grid, flexbox та float хоча б по одному разу.

2.3. Додаткове додавання та ***правильне*** використання різноманітних CSS-технік та фіч буде окремим плюсом в оцінюванні.

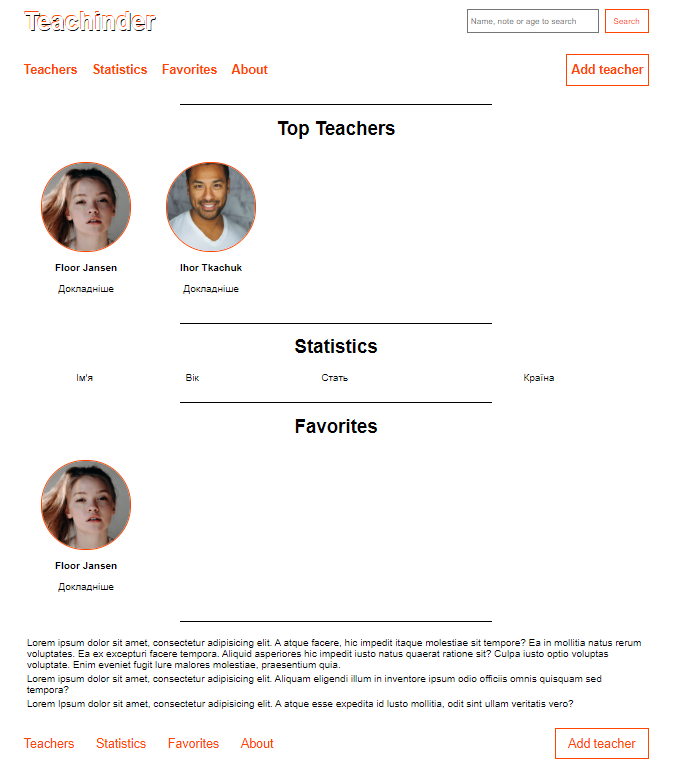
              

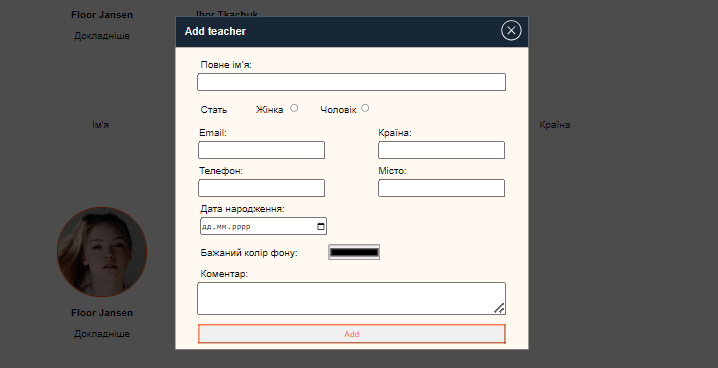
**Завдання 3**. Додати два попапи: перший - для відображення форми додавання викладача; другий - для відображення розширеної інформації про викладача.

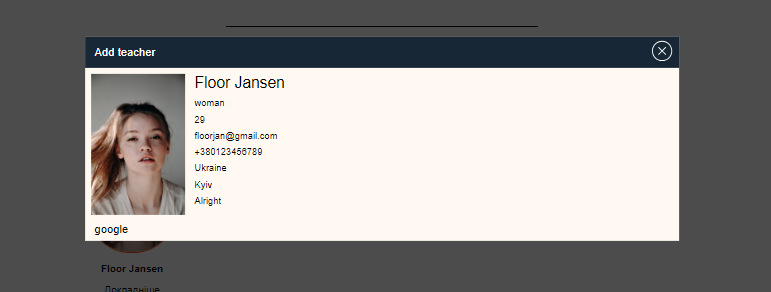
Вони мають бути розміщені по центру екрану.

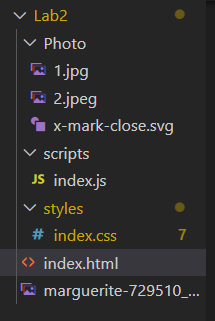
 

**Завдання 4**. За допомогою @media-виразів зробити вигляд сторінки для ширини екрану, меншої за 768 пікселів. Каркас сторінки та розміщення структурних блоків сторінки обирається студентом на власний розсуд.









1. **Висновки до роботи**

За результатами проробленої роботи можемо сказати, що отримали навички створення макету HTML-сторінки з використанням різних технік та познайомилися з технологією CSS.

1. **Контрольні запитання**
2. Типи CSS селекторів

CSS селектори використовуються для вибору та стилізації HTML-елементів на веб-сторінці. Ось деякі з найпоширеніших типів CSS селекторів:

*Елементи селектори* (`element`): Вибирають всі елементи певного типу.

p {

color: blue;

}

*Класові селектори* (`.class`): Вибирають всі елементи з вказаною класом.

.highlight {

background-color: yellow;

}

*Ідентифікаторні селектори* (`#id`): Вибирають єдиний елемент з вказаним ідентифікатором.

#header {

font-size: 24px;

}

*Групові селектори* (`selector1, selector2`): Дозволяють застосовувати однакові стилі до декількох селекторів.

h1, h2 {

font-family: Arial, sans-serif;

}

*Дочірні селектори* (`parent > child`): Вибирають дочірні елементи вкладеного батьківського елемента.

ul > li {

list-style-type: square;

}

*Атрибутні селектори* (`[attribute]`): Вибирають елементи з вказаним атрибутом.

input[type="text"] {

border: 1px solid gray;

}

*Псевдокласи* (`element:pseudo-class`): Вибирають певний стан елемента, наприклад, при наведенні миші (`:hover`), першому дитині (`:first-child`), останньому дитині (`:last-child`), тощо.

a:hover {

color: red;

}

*Псевдоелементи* (`element::pseudo-element`): Додають стилі до певних частин елементів, таких як `::before` і `::after`.

p::before {

content: "»";

}

1. Приоритетність селекторів

При роботі з CSS стилів, селектори можуть конфліктувати, і їхні правила застосовуються з різною пріоритетністю. Це дозволяє вам точно керувати тим, які стилі будуть застосовані до елементів, коли є конфліктні правила. Приоритетність визначається шляхом прирівнювання ваги різних видів селекторів. Ось загальний порядок приоритетності:

1. Вбудовані стилі: Стандартні стилі, які браузер надає за замовчуванням.

2. Селектори елементів та псевдоелементи: Селектори, які використовують імена тегів, такі як `p`, або псевдоелементи, такі як `::before`.

3. Селектори класу, атрибуту та псевдокласи: Селектори з класами (`.class`), атрибутами (`[attribute]`) або псевдокласами (`:hover`).

4. Селектори ідентифікатора: Селектори з ідентифікаторами (`#id`).

5. Селектори стилів інлайн: Стилі, задані безпосередньо в атрибуті `style` елемента HTML.

6. !important: Станів селектора, який має властивість `!important`.

Зазвичай більш специфічні селектори мають вищий пріоритет, але є винятки. Наприклад, якщо ви використовуєте `!important`, це надає властивості максимальну приоритетність, незалежно від інших факторів.

Примір:

p {

color: blue; /\* Пріоритет: 1 \*/

}

.highlight {

color: red; /\* Пріоритет: 2 \*/

}

#special {

color: green; /\* Пріоритет: 3 \*/

}

<p class="highlight" id="special">Цей текст має зелений колір</p>

В багатьох випадках найкращою практикою є уникати використання `!important` та ставити більший акцент на правильне використання специфічних селекторів та організацію коду.

1. Одиниці вимірювання в CSS

У CSS використовуються різні одиниці вимірювання для задання розмірів та відстаней. Ось кілька найпоширеніших одиниць вимірювання:

1. Пікселі (`px`): Піксел - це найменша одиниця вимірювання на екрані. Вказує конкретну кількість пікселів.

font-size: 16px;

width: 300px;

2. Відсотки (`%`): Відсотки відносяться до розміру батьківського елемента. Наприклад, `width: 50%` означає, що елемент буде займати половину ширини свого батьківського елемента.

width: 50%;

height: 75%;

3. ЕМ (`em`): Розмір ем відноситься до розміру шрифту батьківського елемента. Наприклад, якщо розмір шрифту батьківського елемента - 16px, то 1em буде також 16px.

font-size: 1.5em; (1.5 раза більший, ніж розмір шрифту батьківського елемента)

4. Рем (`rem`): Розмір рем відноситься до розміру шрифту кореневого (html) елемента. Це зроблено для більш прогнозованого масштабування стилів на сторінці.

font-size: 1.2rem; (1.2 раза більший, ніж розмір шрифту кореневого елемента)

5. Контейнери (`ch`, `vw`, `vh`, `vmin`, `vmax`): Ці одиниці вимірювання відносяться до розмірів екрана або вікна браузера.

- `ch`: Ширина "0" символу шрифту елемента.

- `vw`: 1% ширини вікна браузера.

- `vh`: 1% висоти вікна браузера.

- `vmin`: Мінімальне значення з `vw` і `vh`.

- `vmax`: Максимальне значення з `vw` і `vh`.

-size: 10ch;

width: 50vw;

height: 25vh;

1. Псевдоелементи та псевдокласи, відмінності

Псевдоелементи та псевдокласи - це часті способи використання CSS для вибору та стилізації певних частин елементів HTML. Однак вони мають трохи різний синтаксис та використання.

*Псевдоелементи* (`::pseudo-element`):

Псевдоелементи дозволяють вам створювати та стилізувати певні частини елементів, які не існують в реальному HTML-структурі. Вони вказуються подвійними двокрапками (`::`) після селектора елемента. Деякі з популярних псевдоелементів:

- `::before`: Додає контент до початку вмісту елемента.

- `::after`: Додає контент після вмісту елемента.

- `::first-line`: Стилізує перший рядок вмісту елемента.

- `::first-letter`: Стилізує першу літеру вмісту елемента.

Приклад використання псевдоелемента:

p::before {

content: "❮";

margin-right: 5px;

}

*Псевдокласи* (`element:pseudo-class`):

Псевдокласи вибирають певні стани або взаємодії елементів. Вони вказуються після селектора елемента та попередньої двокрапки (`:`). Деякі з популярних псевдокласів:

- `:hover`: Стилізує елемент при наведенні миші.

- `:active`: Стилізує елемент під час натискання на нього.

- `:focus`: Стилізує елемент, коли він отримує фокус.

- `:nth-child(n)`: Вибирає елементи з певним порядковим номером.

- `:not(selector)`: Вибирає елементи, які не відповідають вказаному селектору.

Приклад використання псевдокласу:

a:hover {

color: red;

text-decoration: underline;

}

*Відмінності:*

Основна відмінність між псевдоелементами та псевдокласами полягає в тому, що псевдоелементи створюють певні "фіктивні" елементи або частини елементів, які не існують у дереві DOM, тоді як псевдокласи вибирають елементи в залежності від їхнього стану або позиції.

1. Що таке каскад та наслідування

У CSS існують два важливі концепції: каскад та наслідування. Ці концепції допомагають визначити, як стилі застосовуються до елементів на веб-сторінці.

*Каскад* (Cascading):

Каскад у CSS означає, що стилі можуть бути задані на різних рівнях та в різних місцях і мають пріоритетність в залежності від їхньої специфічності та порядку визначення. Якщо різні правила конфліктують, браузер визначає, яке правило застосовувати, враховуючи приоритетність, яка включає в себе приоритетність селекторів та використання `!important`.

*Наслідування* (Inheritance):

Наслідування у CSS означає, що деякі стилі, встановлені для батьківського елемента, можуть автоматично застосовуватися до його дочірніх елементів. Наприклад, якщо батьківський елемент має заданий колір шрифту, то цей колір може успадковуватися дочірніми елементами, якщо для них не встановлено інших стилів.

Це дозволяє зменшити кількість коду та зробити структуру CSS більш логічною. Однак не всі стилі успадковуються. Зазвичай це стосується властивостей, які впливають на відображення тексту, таких як колір, розмір шрифту, наступ і т.д.

1. Боксова модель, види та відмінності

Боксова модель в CSS визначає, як елементи HTML відображаються та займають місце на веб-сторінці. Кожен HTML-елемент можна уявити як прямокутник, внутрішній контент як вміст цього прямокутника, а зовнішній контент як обрамлення цього прямокутника. Це дозволяє точно контролювати розміри та відстані між елементами.

Види боксової моделі:

1. *Content Box* (Бокс вмісту): Включає вміст елемента, який включає текст, зображення та інші внутрішні елементи. Ширина та висота боксу вмісту задається властивостями `width` та `height`.

2. *Padding Box* (Бокс внутрішнього відступу): Охоплює область між вмістом та рамкою елемента. Розмір внутрішнього відступу може бути заданий властивостями `padding`.

3. *Border Box* (Бокс рамки): Включає рамку елемента, яка оточує вміст та внутрішній відступ. Розмір рамки задається властивістю `border`, а товщина рамки - властивістю `border-width`.

4. *Margin Box* (Бокс зовнішнього відступу): Охоплює область за рамкою елемента. Відстань між елементами визначається властивістю `margin`.

Відмінності між типами боксової моделі:

Один з головних аспектів відмінностей між типами боксової моделі полягає в тому, як властивості `width` та `height` впливають на фактичний розмір елемента. В основі боксової моделі лежить різниця між заданими розмірами та зовнішніми розмірами елемента.

- Content Box: Розмір елемента включає тільки вміст, без врахування внутрішнього відступу, рамки та зовнішнього відступу.

- Border Box: Розмір елемента включає вміст, внутрішній відступ, рамку, але не зовнішній відступ.

Це має значення при розрахунках розмірів елементів, особливо коли застосовуються внутрішні відступи та рамки, а також в контексті адаптивного дизайну та вирівнювання елементів. Вибір типу боксової моделі залежить від ваших потреб та специфіки проекту.

1. Що таке margin, padding, border, як працює margin collapse

`margin`, `padding` та `border` - це важливі концепції в CSS, які визначають розміри та зовнішній вигляд елементів HTML. Давайте розглянемо кожну з цих концепцій:

1. *Margin* (Зовнішній відступ):

`margin` - це простір ззовні від рамки елемента. Він визначає відстань між елементом та іншими елементами на сторінці. Наприклад:

margin: 10px;

margin-top: 10px;

margin-right: 20px;

margin-bottom: 15px;

margin-left: 5px;

2. *Padding* (Внутрішній відступ):

`padding` - це простір вздовж внутрішньої границі елемента, між рамкою та вмістом. Він використовується для визначення відстані між вмістом елемента та його рамкою:

padding: 10px;

padding-top: 10px;

padding-right: 20px;

padding-bottom: 15px;

padding-left: 5px;

3. *Border* (Рамка):

`border` - це границя навколо вмісту елемента. Вона може мати різну товщину, стиль та колір:

border: 1px solid black;

border-width: 2px;

border-style: dashed;

border-color: red;

Тепер щодо `margin collapse` (злиття зовнішнього відступу):

`margin collapse` - це явище, коли зовнішні відступи двох сусідніх блоків зливаються разом, формуючи одну велику відстань, а не суму відстаней кожного відступу окремо. Це відбувається, коли верхній або нижній зовнішні відступ одного блока перетинається з верхнім або нижнім зовнішнім відступом іншого блока.

Це може бути непередбачуваним результатом, коли працюєте з розміщенням елементів. Щоб уникнути цього, можна використовувати внутрішні відступи (`padding`) або встановлювати `overflow: hidden;` для контейнерів, які зазнають проблеми з злиттям зовнішніх відступів.

Злиття зовнішніх відступів може бути навіть корисним в деяких ситуаціях, але варто розуміти, коли це може виникнути та як це може вплинути на розміщення контенту.

1. Поток документу, як працює float та clear

Потік документу - це спосіб, яким HTML-елементи відображаються на веб-сторінці у визначеному порядку. За замовчуванням, елементи відображаються один за одним в потоці, від верху до низу сторінки. Однак, властивості `float` та `clear` дозволяють вам змінювати цей потік для досягнення певних дизайнерських ефектів.

*Float* (Плавання):

Властивість `float` дозволяє елементам "плавати" ліворуч або праворуч від батьківського елемента або інших елементів. Це використовується для створення зміщених колонок, обтікання зображень текстом та багато інших макетів.

Наприклад, якщо ви хочете, щоб два блока плавали один поруч одного, ви можете використовувати такий код:

.left {

float: left;

width: 50%;

}

.right {

float: right;

width: 50%;

}

*Clear* (Очищення):

Властивість `clear` використовується для визначення, як елемент повинен поводитися щодо зміщених елементів, які використовують `float`. Вона вказує, які сторони елемента повинні бути "очищені" від зміщених елементів. Значення `clear` може бути `left`, `right`, `both` (обидві сторони) або `none` (без очищення).

.clear-left {

clear: left;

}

.clear-right {

clear: right;

}

З використанням `float` можуть виникати певні проблеми, такі як проблеми з обтіканням тексту, розірванням потоку та інші. Тому, коли ви використовуєте `float`, важливо розуміти, як він впливає на розміщення елементів та враховувати це при проектуванні та розробці веб-сайту.

1. Позиціонування, z-index

Позиціонування та властивість `z-index` в CSS дозволяють вам контролювати розміщення елементів на веб-сторінці, включаючи їхній порядок накладання.

Позиціонування:

1. Static (Статичний): Це значення за замовчуванням. Елементи відображаються в потоці документа без будь-якого спеціального позиціонування.

2. Relative (Відносний): Властивості `top`, `right`, `bottom` та `left` визначають відстань, на яку елемент буде зсунутий від своєї звичайної позиції в потоці документа.

3. Absolute (Абсолютний): Елемент видаляється з потоку документа і позиціонується відносно батьківського елемента, який має встановлену властивість `position: relative` або інші позиціонування (крім `static`).

4. Fixed (Фіксований): Елемент закріплюється на екрані і не змінює своє положення, коли сторінка прокручується. Він позиціонується відносно вікна браузера.

5. Sticky (Приклеєний): Елемент веде себе як `relative` до певної позиції прокручування, після чого він стає `fixed` відносно батьківського контейнера.

z-index (З індекс):

`z-index` визначає порядок накладання елементів, коли вони мають позицію, відмінну від `static`. Значення `z-index` вказує, який елемент повинен бути вище чи нижче в порядку накладання.

Більше значення `z-index` вище - елемент буде ближче до переднього плану. Зауважте, що `z-index` працює тільки для елементів, які мають встановлені позиції `relative`, `absolute` або `fixed`.

Приклад:

.parent {

position: relative;

z-index: 1;

}

.child {

position: absolute;

z-index: 2;

}

В даному випадку, `.child` буде вище за `z-index`, тобто він буде накладатися на `.parent`.

1. Сутність Flexbox, основні принципи та CSS-правила

Flexbox (або гнучкий контейнер) - це модель розміщення у CSS, яка дозволяє легко створювати адаптивні макети та вирівнювання елементів всередині контейнера, незалежно від їхнього розміру чи порядку.

Основні принципи Flexbox:

1. *Контейнер та його діти*: Виберіть контейнер (батьківський елемент) та встановіть йому `display: flex;`, щоб вказати, що цей контейнер використовує модель Flexbox. Всі діти контейнера стають гнучкими елементами.

2. *Ось гнучкості*: Flexbox працює на осі, яка може бути горизонтальною (рядковою) або вертикальною (колонковою). Виберіть ось гнучкості за допомогою властивості `flex-direction: row;` (горизонтальна) або `flex-direction: column;` (вертикальна).

3. *Вирівнювання по осі*: Властивості `justify-content` та `align-items` дозволяють вирівнювати елементи вздовж головної та перпендикулярної осей в контейнері відповідно.

4. *Розтягнення та зжимання*: Flexbox автоматично розтягує та зжимає елементи відповідно до їхнього розміру та доступного простору. Ви можете контролювати це за допомогою властивостей `flex-grow`, `flex-shrink` та `flex-basis`.

5. *Вирівнювання всередині контейнера*: Властивість `align-content` використовується для вирівнювання рядків у випадку, коли їхніх не вистачає для заповнення всього контейнера.

Основні CSS-правила Flexbox:

.container {

display: flex;

flex-direction: row; /\* або column \*/

justify-content: center; /\* горизонтальне вирівнювання \*/

align-items: center; /\* вертикальне вирівнювання \*/

align-content: center; /\* вирівнювання рядків \*/

}

.item {

flex: 1; /\* розтягнення/зжимання елемента \*/

}

Flexbox дозволяє створювати складні макети з мінімальною кількістю CSS-коду та відсутністю необхідності в багатошарових розміщеннях, злиттях та інших нюансах позиціонування. Він допомагає створити гнучкі та адаптивні макети, які виглядають добре на різних пристроях та екранних розмірах.

1. Сутність Grid, основні принципи та CSS-правила

CSS Grid (сітка) - це потужний механізм розміщення елементів на веб-сторінці у вигляді двовимірної сітки. Він надає велику гнучкість та контроль над розміщенням елементів, дозволяючи створювати складні макети з легкістю.

Основні принципи CSS Grid:

1. *Контейнер та його сітка*: Визначте контейнер, у якому ви хочете використовувати сітку, за допомогою `display: grid;`. Ви також можете визначити розмір колонок та рядків за допомогою `grid-template-columns` та `grid-template-rows`.

2. *Діти та їх позиціонування*: Додайте дітей в контейнер та визначте, як вони мають розміщуватися на сітці. Використовуйте `grid-column` та `grid-row` для визначення позицій елементів.

3. *Grid Lines* (Сіткові лінії): Сіткові лінії - це вертикальні та горизонтальні лінії, які утворюють сітку. Ви можете визначати властивості `grid-column` та `grid-row` за допомогою номерів сіткових ліній.

4. *Grid Gaps* (Відступи сітки): Встановіть відступи між колонками та рядками за допомогою `grid-column-gap` та `grid-row-gap`. Також ви можете використовувати `grid-gap` для визначення обох видів відступів одразу.

5. *Grid Areas* (Області сітки): Визначте зони, які займають кілька колонок та рядків, за допомогою `grid-template-areas`. Це дозволяє створювати складні макети за допомогою назв областей.

Основні CSS-правила CSS Grid:

.container {

display: grid;

grid-template-columns: 1fr 1fr 1fr; /\* три однакові колонки \*/

grid-template-rows: auto auto; /\* дві автоматично розтягуючі рядки \*/

grid-gap: 10px; /\* відступи між елементами \*/

grid-template-areas:

"header header header"

"sidebar main main"

"footer footer footer"; /\* зони сітки \*/

}

.header {

grid-area: header;

}

.sidebar {

grid-area: sidebar;

}

.main {

grid-area: main;

}

.footer {

grid-area: footer;

}

CSS Grid надає більше можливостей для створення різноманітних макетів, дозволяючи створювати як прості, так і дуже складні структури. Він особливо корисний для вирішення задач зі складним розміщенням елементів, таких як адаптивний дизайн, багатоколонкові макети та більш складні веб-сторінки.

1. Responsive, основні принципи побудови сторінки

Адаптивний (респонсивний) дизайн веб-сторінки означає, що сторінка відповідає на різні розміри екранів та пристроїв, забезпечуючи користувачам оптимальний досвід перегляду незалежно від того, на чому вони переглядають вміст.

Основні принципи адаптивного дизайну:

1. *Гнучка сітка* (Flexible Grid): Використовуйте сітку (Flexbox або CSS Grid), яка дозволяє розміщувати елементи у гнучкому режимі, адаптуючись до різних розмірів екранів.

2. *Медіа-запити* (Media Queries): Використовуйте медіа-запити для визначення розмірів екранів, на яких потрібно застосовувати різні стилі. Наприклад, ви можете змінювати розмір шрифтів, колонок сітки та інші стилі на певних розмірах екранів.

3. *Відносний розмір* (Relative Sizing): Встановлюйте розміри елементів, відступи та інші параметри відносно відсотків або em/rem, замість абсолютних одиниць вимірювання.

4. *Зображення та медіа*: Використовуйте властивості `max-width: 100%;` та `height: auto;` для зображень та медіа, щоб вони пропорційно масштабувались на різних екранах.

5. *Мобільна перша* (Mobile-First): Розробляйте з думкою про мобільні пристрої, а потім додаючи стилі для більших екранів. Це допомагає забезпечити базовий функціонал та доступність на всіх пристроях.

6*. Тести та аналіз*: Перевіряйте дизайн на різних пристроях та розмірах екранів, використовуючи інструменти для аналізу адаптивності. Відстежуйте проблеми та коригуйте їх для кращого користувацького досвіду.

7. *Збереження функціональності*: Впевніться, що важливі функції та інформація доступні на всіх розмірах екранів, адже деякі функції можуть бути приховані на менших пристроях через обмежений простір.

1. @media-вирази, <meta name=”viewport” />

`@media` вирази та `<meta name="viewport" />` - це важливі аспекти адаптивного дизайну, які допомагають налаштувати поведінку та вигляд веб-сторінки на різних пристроях.

`@media` вирази:

Медіа-запити (`@media` вирази) дозволяють змінювати стилі в залежності від характеристик пристрою, таких як розмір екрану, орієнтація, роздільна здатність тощо. Зазвичай їх використовують у CSS для забезпечення адаптивності дизайну.

Приклад медіа-запиту для зміни стилів на екранах шириною менше 600px:

@media (max-width: 600px) {

/\* Ваші стилі тут \*/

}

`<meta name="viewport" />` - це метатег HTML, який використовується для налаштування параметрів відображення на мобільних пристроях. Він допомагає браузеру зрозуміти, як правильно масштабувати та відображати вміст на різних розмірах екранів.

Приклад використання `<meta name="viewport" />`:

<!DOCTYPE html>

<html>

<head>

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

</head>

<body>

<!-- Ваш вміст тут -->

</body>

</html>

У даному прикладі, `width=device-width` вказує браузеру використовувати фактичну ширину пристрою як ширину відображення, а `initial-scale=1.0` встановлює початковий масштаб в 100%.