**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №5**

«JavaScript. «Події. Регулярні вирази»»

з дисципліни  
«Основи Front-end технологій»

Виконала:

студентка групи ІМ-11   
Бащак Ярина Володимирівна

варіант 1

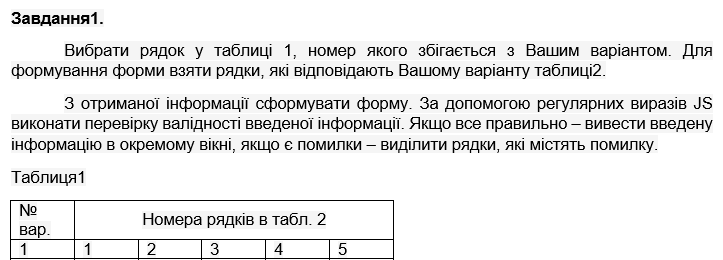
Перевірив:

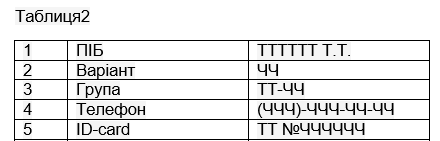
доц. Жереб К.А.

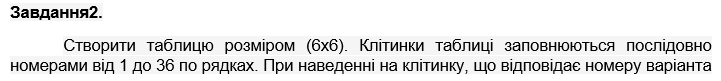
Київ 2023

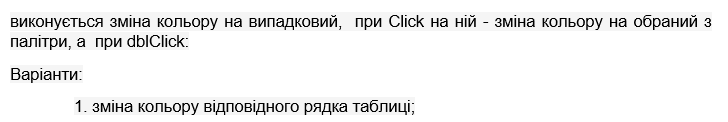
**Завдання.**

Мій варіант №1.

****

****

****

****

**Хід роботи.**

Завдання 1:

*Що є в HTML.*

Для створення форми я використала тег form, для створення полів вводу input з атрибутом type="text" або "number". Також вирішила додати підказки під кожним полем вводу, які показуються, якщо введені дані неправильного формату. Наприклад,



Цього було досягнуто за допомогою додаткового тега p після кожного input, якому додаються/віднімаються певні css-класи, щоб керувати його виглядом (про класи пізніше).

<form id="myForm">

    <p id="form-title">Будь ласка, заповніть форму</p>

    <input placeholder="ПІБ" type="text" id="name" name="name"><br>

    <p id="name-tip" class="tip">Формат: Прізвище І.П.</p>

    <input placeholder="Варіант" type="number" id="variant" name="variant"><br>

    <p id="variant-tip" class="tip">Формат: Ч або ЧЧ</p>

    <input placeholder="Група" type="text" id="group" name="group"><br>

    <p id="group-tip" class="tip">Формат: ТТ-ЧЧ</p>

    <input placeholder="Телефон" type="text" id="phone" name="phone"><br>

    <p id="phone-tip" class="tip">Формат: (ЧЧЧ)-ЧЧЧ-ЧЧ-ЧЧ</p>

    <input placeholder="ID-card" type="text" id="id-card" name="id-card"><br>

    <p id="id-card-tip" class="tip">Формат: ТТ №ЧЧЧЧЧЧ</p>

    <input type="button" value="Відправити" onclick="validateForm()">

</form>

Як видно, кожна підказка має такий самий ідентифікатор, як у поля вводу, до якого вона відноситься, але з додаванням “-tip”. В майбутньому це полегшить доступ до цих елементів з JS.

*Класи в css, які допомагають візуально виділити неправильно заповнені поля.*

Щоб придати полю вигляду помилки або показати підказку з форматом, я додаю їм css-клас error. Для тега input клас error робить червоний і товстіший контур, також зафарбовує в червоний текст за замовчуванням (placeholder), якщо в поле нічого не було введено.

input.error {

  border: 2px solid rgb(235, 119, 119);

}

input.error::placeholder {

  color: rgb(236, 92, 92);

}

А для тегу p клас error робить підказку видимою (за замовчуванням ця підказка червона, але не видима за рахунок display: none).

.tip {

  color: rgb(236, 92, 92);

  display: none;

}

.tip.error {

  display: inline;

}

*Як реалізована перевірка.*

За валідацію форми відповідає функція validateForm(), яка викликається при натисканні на кнопку «Відправити».

Для зручності ідентифікатори кожного поля я помістила в масив, а регулярні вирази - в окремий об’єкт.

const fields = ['name', 'variant', 'group', 'phone', 'id-card']

const ukChar = '[А-ЩЬЮЯҐЄІЇа-щьюяґєії]'

const regex = {

  name: new RegExp(`^${ukChar}+ ${ukChar}\\.${ukChar}\\.$`),

  variant: /^\d{1,2}$/,

  group: new RegExp(`^${ukChar}{2}-\\d{2}$`),

  phone: /^\(\d{3}\)-\d{3}-\d{2}-\d{2}$/,

  'id-card': new RegExp(`^${ukChar}{2} №\\d{6}$`)

}

На початку функції створюю пусті масиви inputs і errors для збереження введених даних по кожному полю і ідентифікаторів полів з неправильно даними відповідно. Потім проходжусь по кожному значенню з масиву fields, зчитую значення відповідного поля, прибираю css-клас error з поля і з тексту-підказки і перевіряю чи введені дані відповідають шаблону. Якщо ні - додаю цей ідентифікатор у масив errors.

const removeError = id => {

  document.getElementById(id).classList.remove('error')

  document.getElementById(`${id}-tip`).classList.remove('error')

}

function validateForm () {

  const inputs = {}

  const errors = []

  fields.forEach(id => {

    inputs[id] = document.getElementById(id).value

    removeError(id)

    if (!regex[id].test(inputs[id])) {

      errors.push(id)

    }

  }) …

Далі якщо масив errors не пустий, проходжусь по ньому додаючи відповідним полю і підказці клас error, а якщо пустий, тобто всі дані правильні, - виводжу їх у спливаючому вікні за допомогою функції alert(). Але була проблема, що хоча alert викликається пізніше ніж removeError, який видаляє всі класи візуальних помилок, alert блокує основний потік, в тому числі візуальні оновлення, які не синхронізовані з виконанням коду (браузер може відкласти візуальне відтворення змін на «пізніше»). Як наслідок, візуальні зміни, які були внесені до alert, відображалися лише після закриття вікна сповіщення. З метою це обійти alert додатково огорнутий в setTimeout з таймером 0 мс – це вирішує проблему.

…

  if (errors.length > 0) {

    errors.forEach(id => {

      document.getElementById(id).classList.add('error')

      document.getElementById(`${id}-tip`).classList.add('error')

    })

  } else {

    setTimeout(() => {

      alert(

        `ПІБ: ${inputs['name']}\nВаріант: ${inputs['variant']}\nГрупа: ${inputs['group']}\nТелефон: ${inputs['phone']}\nID-card: ${inputs['id-card']}`

      )

    }, 0)

  }

}

Завдання 2:

В HTML створюю палітру, щоб можна було вибирати колір, і пусту таблицю.

    <input type="color" id="color-picker" value="#4E4FFF">

    <table id="table"></table>

Далі, щоб заповнити цю таблицю потрібними клітинками, в js-файлі роблю

1. цикл, який створює рядки за допомогою document.createElement('tr')
2. цикл, який створює клітинки в кожному з рядків за допомогою document.createElement('td')

const squareSize = 6

const table = document.getElementById('table')

for (let i = 0; i < squareSize; i++) {

  const row = document.createElement('tr')

  for (let j = 0; j < squareSize; j++) {

    const cell = document.createElement('td')

    cell.id = cell.textContent = squareSize \* i + j + 1

    row.appendChild(cell)

  }

  table.appendChild(row)

}

і в кінці кожного додаю їх у батьківський елемент за допомогою appendChild.

Далі пишу функцію changeColor для зафарбовування однієї клітинки, яка в якості параметрів приймає клітинку і колір, в який її потрібно зафарбувати. Колір вказувати не обов’язково, він за замовчуванням дорівнює null. Якщо колір не був переданий - він визначається випадковим чином за допомогою додаткової функції getRandomColor.

const getRandomColor = () =>

  '#' + Math.floor(Math.random() \* 16777215).toString(16)

const changeColor = (cell, color = null) => {

  if (color === null) {

    color = getRandomColor()

  }

  cell.style.backgroundColor = color

}

Тепер можна додати потрібні події на клітинку. Якщо є наведення на клітинку (mouseover), то функція changeColor без вказання кольору, якщо клік (click) – з вказаним кольором, а якщо подвійний клік (dblclick) – знаходження батьківського елементу для клітинки за допомогою parentNode і цикл, який проходить по кожній клітинці знайденого рядка і змінює колір на вказаний.

const variant = 1

const myCell = document.getElementById(variant)

myCell.addEventListener('mouseover', () => changeColor(myCell))

myCell.addEventListener('click', () => changeColor(myCell, selectedColor))

myCell.addEventListener('dblclick', () => {

  const row = myCell.parentNode

  for (cell of row.cells) {

    changeColor(cell, selectedColor)

  }

})

Вказаний колір ми отримуємо теж за допомогою доданої події, але на елемент, який представляє палітру, і з типом події введення (input).

const colorPicker = document.getElementById('color-picker')

let selectedColor = 'rgb(78, 79, 254)'

colorPicker.addEventListener('input', () => {

  selectedColor = colorPicker.value

})

**Результати:**

[Посилання на репозиторій в Github.](https://github.com/yaryna-bashchak/basics-of-frontend/tree/main/lab5)

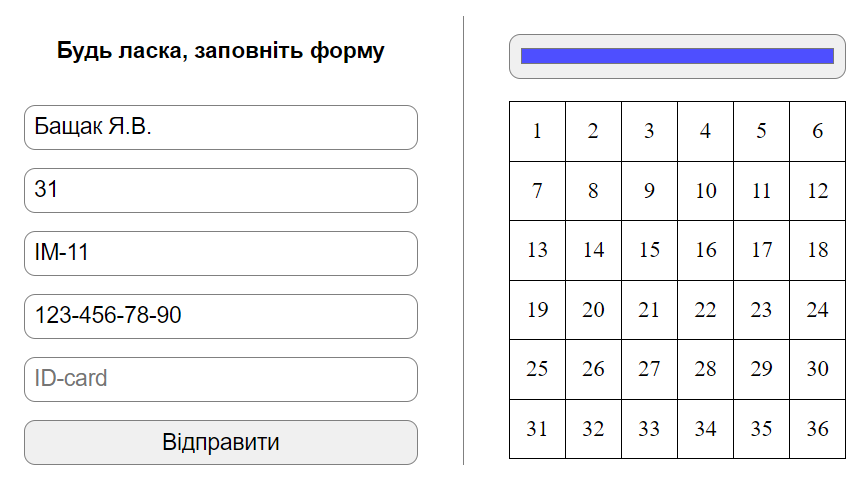
[Посилання на сайт.](https://yaryna-bashchak.github.io/basics-of-frontend/lab5/index.html)

Щоб запустити в себе цю роботу, достаньо відкрити файл index.html у своєму браузері.

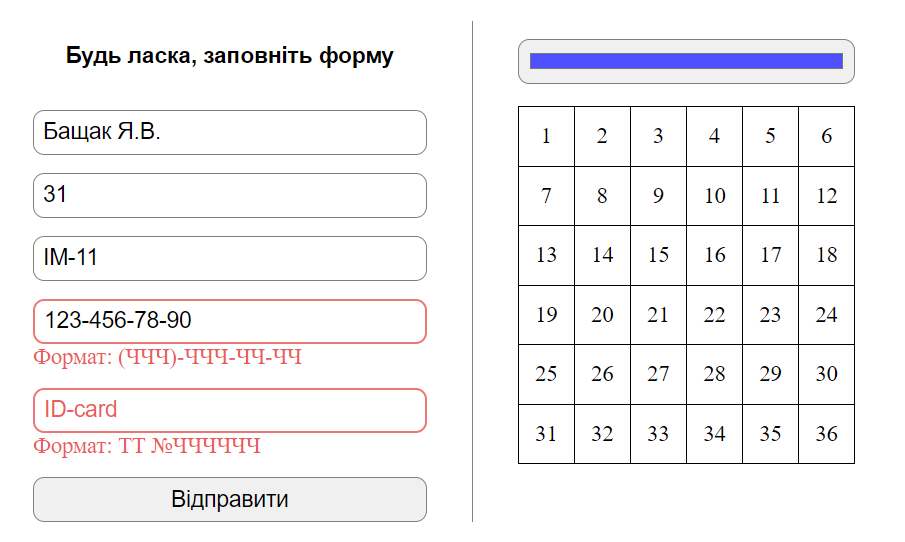
Скріншоти сайту:

*Завдання 1:*

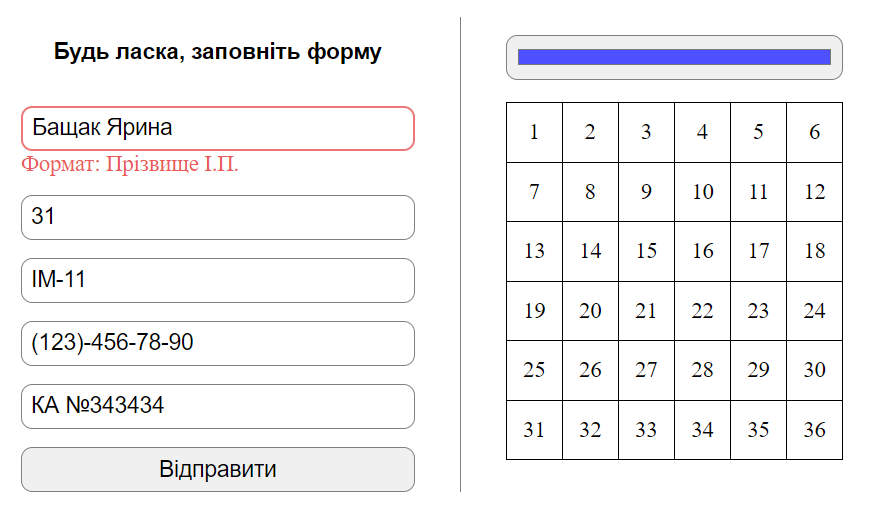
* до відправки форми



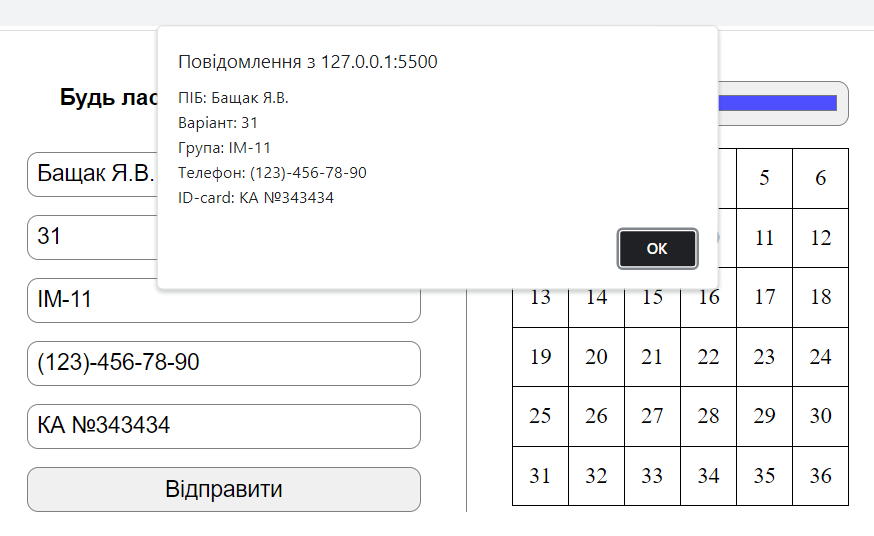
* після відправки (загоряються неправильно введені поля)



* якщо неправильні тепер заповнити правильно, але припуститися помилки в іншому і відправити (всі попередні зникають, підсвічуються нові)

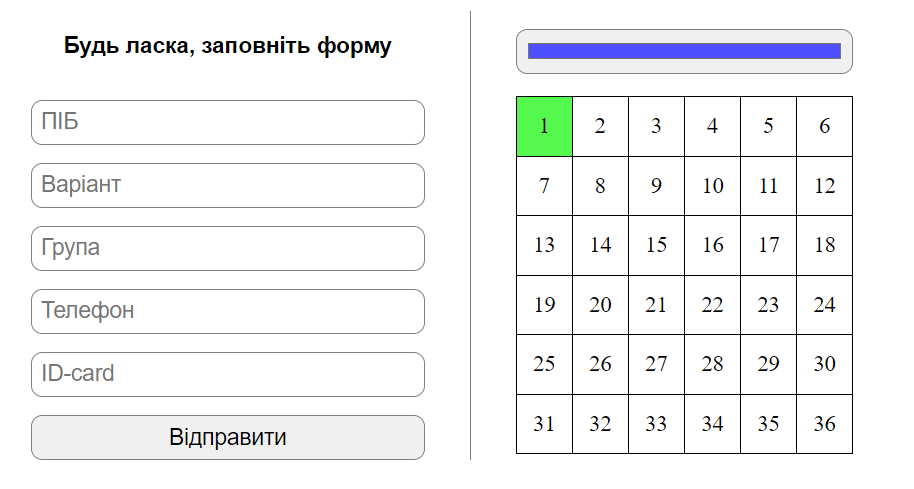


* якщо заповнити всі поля правильно і відправити, то появляється вікно з введеною інформацією, а всі помилки зникають

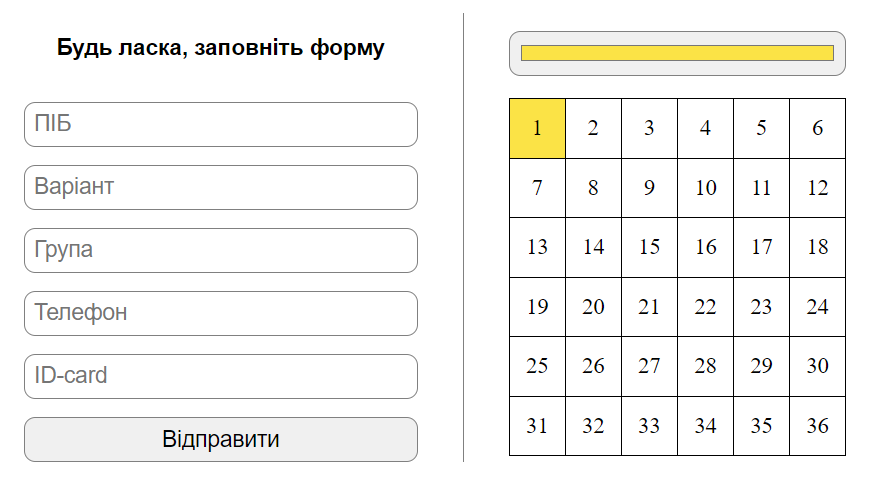


*Завдання 2:*

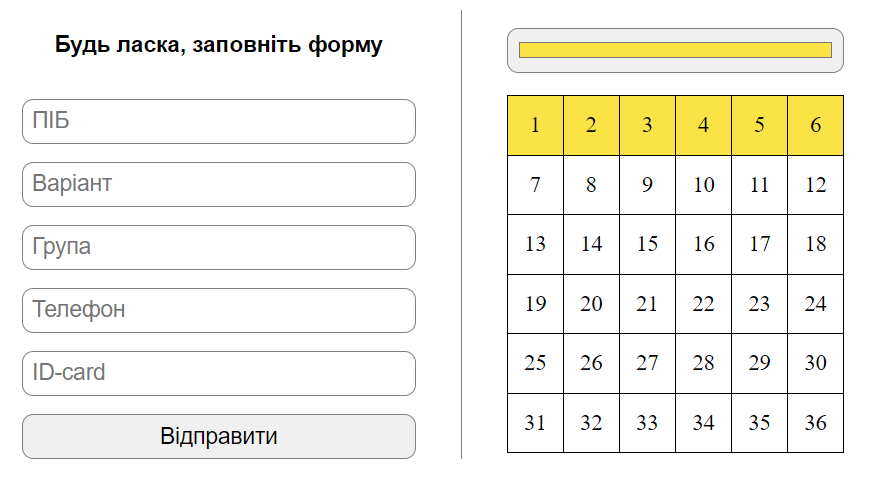
* наведення на клітинку №1



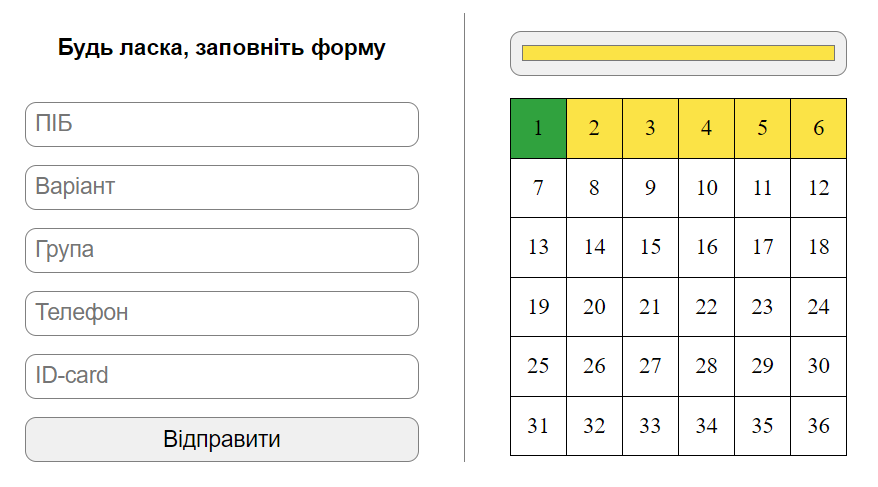
* вибір нового кольору + клік



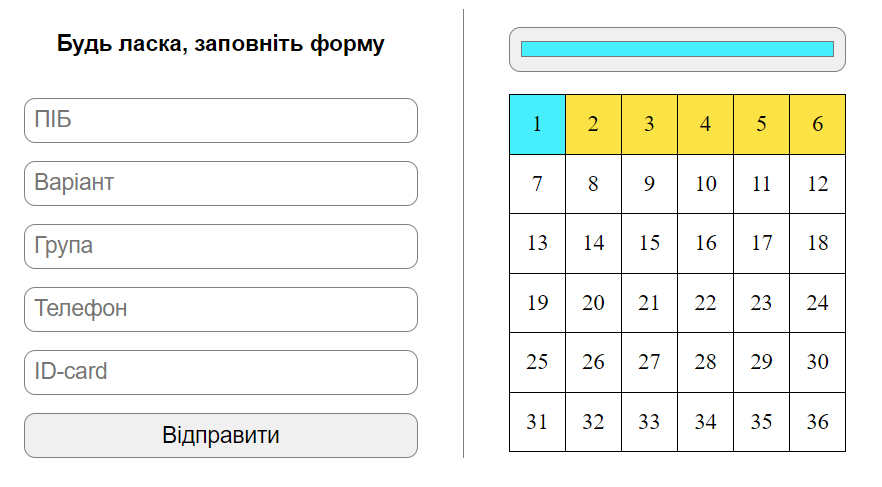
* подвійний клік



* ще одне наведення на клітинку



* ще один вибір нового кольору + клік



Висновок: Під час цієї лабораторної роботи я ще раз попрактикувалася у доступі до html-елементів, вносячи певні зміни у їх візуальну складову, а також створила декілька вкласних подій, які реагують на різні типи.