МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних систем та технологій

**Лабораторна робота № 3**

з дисципліни **«**Технології програмування**»**

**Тема: «Функції та методи функцій»**

**Виконав студент 2-го курсу**

Микитенко Ілля Андрійович

Група: ІР-21

Варіант 11

**Перевірила:**
Бондаренко Ольга Сергіївна

Київ – 2025

**Завдання 1**

Написати функцію, яка приймає число і  повертає «додатне», «від’ємне» або «нуль»

**Код програми:**

```js
function checkNumber(num) {
  const number = Number(num);


  if (number > 0) {
    return "Додатнє";
  } else if (number < 0) {
    return "Від'ємне";
  } else if (number === 0) {
    return "Нуль";
  } else {
    return "Не є числом";
  }
}

const numberInput = document.getElementById('numberInput');
const checkButton = document.getElementById('checkButton');
const resultSpan = document.getElementById('resultSpan');

checkButton.addEventListener('click', () => {
  const inputValue = numberInput.value;
  const result = checkNumber(inputValue);
  resultSpan.textContent = result;
});
```

```html
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Перевірка числа</title>
</head>
<body>
    <h2>Перевірка числа</h2>
    <p>Введіть число, щоб дізнатись, чи воно додатне, від'ємне або нуль.</p>

    <input id="numberInput" placeholder="Введіть число">
    <button id="checkButton">Перевірити</button>

    <p><strong>Результат:</strong> <span id="resultSpan"></span></p>

    <script src="laba 3.1.js"></script>
</body>
</html>
```

**Приклад виконання:**

---

# Перевірка числа

Введіть число, щоб дізнатись, чи воно додатне, від'ємне або нуль.

5 Перевірити

**Результат:** Додатнє

**Завдання 2**

Згенерувати масив із 12 випадкових чисел. Вивести індекси та значення елементів, а також окремо всі від’ємні.

**Код програми:**

```
let arr = [];
for (let i = 0; i < 12; i++) {
    let randomNumber = Math.floor(Math.random() * 101) - 50;
    arr.push(randomNumber);
}
console.log("Початковий масив:", arr);

console.log("\nВсі елементи масиву:");
for (let i = 0; i < arr.length; i++) {
    console.log(`Індекс: ${i}, Значення: ${arr[i]}`);
}

console.log("\nТільки від'ємні елементи:");
for (let i = 0; i < arr.length; i++) {
    if (arr[i] < 0) {
        console.log(arr[i]);
    }
}
```

**Приклад виконання:**

```
[Running] node "c:\Users\крутой пацан 13\VSCODE\laba 3.2.js"
Початковий масив: [
  -21,  43, -35,  41, 32,
   49, -25, -37, -50, 25,
   32,  33
]

Всі елементи масиву:
Індекс: 0, Значення: -21
Індекс: 1, Значення: 43
Індекс: 2, Значення: -35
Індекс: 3, Значення: 41
Індекс: 4, Значення: 32
Індекс: 5, Значення: 49
Індекс: 6, Значення: -25
Індекс: 7, Значення: -37
Індекс: 8, Значення: -50
Індекс: 9, Значення: 25
Індекс: 10, Значення: 32
Індекс: 11, Значення: 33

Тільки від'ємні елементи:
-21
-35
-25
-37
-50

[Done] exited with code=0 in 0.105 seconds
```

**Контрольні питання:**

**1. Що таке функція в JavaScript?**

Функція в **JavaScript** — це блок коду, який можна викликати (виконати) та використовувати багаторазово. Вона дозволяє організувати код, уникнути повторень та зробити його більш читабельним. Функції можуть приймати вхідні дані (параметри) та повертати результат. За своєю суттю, функція — це особливий тип об'єкта.

**2. Способи оголошення функцій та їх відмінності**

У **JavaScript** є кілька основних способів оголошення функцій:

- **Function Declaration (Оголошення функції):**

JavaScript

```
function sayHello() {
  console.log('Hello!');
}
```

  - **Відмінність:** Цей спосіб підтримує **hoisting** (підняття). Це означає, що функцію можна викликати навіть до її оголошення у коді, оскільки інтерпретатор **JavaScript** "піднімає" її у пам'яті на етапі компіляції.
- **Function Expression (Функціональний вираз):**

JavaScript

```
const sayHello = function() {
  console.log('Hello!');
};
```

  - **Відмінність:** Цей спосіб **не підтримує hoisting**. Функцію можна викликати лише після того, як вона була оголошена. Це пов'язано з тим, що вона присвоюється змінній, а змінні також піднімаються, але їхнє значення (у даному випадку, функція) ініціалізується лише в місці оголошення.

**3. Що таке стрілкова функція?**

**Стрілкова функція** (arrow function) — це більш стислий синтаксис для написання функціональних виразів.

JavaScript

```
const sayHello = () => {
  console.log('Hello!');
};
```

- **Переваги:**

    - **Короткий синтаксис:** Особливо зручно для простих, однорядкових функцій.
    - **Відсутність власного this:** Стрілкові функції не створюють свій власний контекст this. Натомість, вони успадковують this від батьківської (зовнішньої) області видимості. Це вирішує поширені проблеми з this у звичайних функціях.
- **Обмеження:**
    - **Не мають arguments:** Не мають доступу до об'єкта arguments.
    - **Не можна використовувати як конструктор:** Не можуть бути викликані з new.
    - **Не підходять для методів об'єктів:** Через успадкування this від зовнішньої області, вони можуть працювати неочікувано як методи об'єктів.

**4. «Функції – об’єкти першого класу»**

Поняття **"функції — об’єкти першого класу"** (first-class citizens) означає, що в **JavaScript** функції мають ті ж права, що й інші змінні:

- Їх можна присвоювати змінним.
- Їх можна передавати як аргументи в інші функції.
- Їх можна повертати з інших функцій.
- Їх можна зберігати в масивах та об'єктах.

**5. Параметри та аргументи функцій**

- **Параметри (parameters)** — це змінні, які оголошуються в дужках при визначенні функції. Вони є «заповнювачами» для значень.
- **Аргументи (arguments)** — це конкретні значення, які передаються функції під час її виклику.

**Приклад:**

JavaScript

```
function greet(name) { // name - це параметр

  console.log(`Hello, ${name}!`);

}

greet('Alex'); // 'Alex' - це аргумент
```

**Перевірка типів аргументів:** У **JavaScript** немає вбудованої суворої перевірки типів. Для цього можна використовувати:

- **Оператор typeof:** Перевіряє тип аргументу. Наприклад: if (typeof name !== 'string') { ... }.

- **Додаткові бібліотеки:** Наприклад, **TypeScript** або **JSDoc**, які додають статичну типізацію та дозволяють виявляти помилки на ранніх етапах розробки.

**6. Область видимості змінних (Scope)**

**Область видимості** — це доступність змінних, функцій та об'єктів у певних частинах коду.

- **Глобальна область (Global Scope):** Змінні, оголошені поза будь-якими функціями чи блоками, доступні з будь-якої точки програми.
- **Функціональна область (Function Scope):** Змінні, оголошені всередині функції (з використанням var), доступні лише всередині цієї функції.
- **Блочна область (Block Scope):** Змінні, оголошені з let та const усередині блоків коду (наприклад, if, for, {}), доступні лише в межах цього блоку.

**7. Що таке замикання (Closure)?**

**Замикання** — це функція, яка "пам'ятає" та має доступ до змінних зі своєї зовнішньої області видимості, навіть після того, як ця зовнішня функція завершила своє виконання.

**Приклад:**

JavaScript

```
function createCounter() {
  let count = 0; // Змінна з зовнішньої області видимості
  return function() { // Внутрішня функція-замикання
    count++;
    return count;
  };
}
const counter = createCounter();
console.log(counter()); // Виведе 1
console.log(counter()); // Виведе 2
```

В цьому прикладі counter — це внутрішня функція, яка продовжує мати доступ до змінної count навіть після завершення роботи createCounter.

**8. Що таке рекурсія?**

**Рекурсія** — це підхід до вирішення задачі, коли функція викликає саму себе, щоб розбити велику задачу на менші, подібні до неї підзадачі. Важливо мати базовий випадок (умова виходу), щоб уникнути нескінченного циклу.

- **Переваги:**
  - **Елегантність:** Дозволяє писати лаконічний і зрозумілий код для розв'язання певних задач, наприклад, обходу дерева або обчислення факторіалу.
- **Недоліки:**
  - **Складність відстеження:** Може бути важко відстежити потік виконання.
  - **Витрати пам'яті:** Кожен рекурсивний виклик додається до стека викликів, що може призвести до його переповнення (**"stack overflow"**) для глибоких рекурсій.

**9. Методи роботи з масивами**

- **forEach():** Перебирає кожен елемент масиву, виконуючи для нього вказану функцію. **Не повертає** новий масив.
- **map():** Перебирає кожен елемент і **повертає новий масив**, що складається з результатів виконання функції для кожного елемента.
- **filter(): Повертає новий масив**, що містить лише ті елементи, для яких вказана функція-умова повернула true.
- **reduce():** Зводить (редукує) масив до єдиного значення. Приймає два аргументи: функцію-акумулятор та початкове значення.

**10. Різниця між call, apply та bind**

Всі ці методи змінюють контекст this для функції.

- **call():** Викликає функцію **одразу**, передаючи аргументи **окремим списком**.

JavaScript

myFunc.call(thisArg, arg1, arg2, ...);

- **apply():** Викликає функцію **одразу**, передаючи аргументи **як масив (або схожий на масив об'єкт)**.

JavaScript

myFunc.apply(thisArg, [arg1, arg2, ...]);

- **bind(): Не викликає** функцію одразу. Натомість, він **повертає нову функцію**, яка має постійно прив'язаний контекст this. Цю нову функцію можна викликати пізніше.

JavaScript

```
const newFunc = myFunc.bind(thisArg);

newFunc(); // Викликаємо пізніше
```