**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**ЛАБОРАТОРНА РОБОТА № 9**

**Тема**: “Використання стандартних та розроблених методів у програмах.”

Виконав студент І курсу

спеціальності «Основи веб-розробки »

Вялков Владислав Костянтинович

**Ужгород-2024**

Мета: В освоєнні основних методів створення та відправки веб-форм, використовуючи HTML, CSS та JavaScript, а також вивченні роботи з GET та POST запитами. Ви будете досліджувати різні типи елементів форм і навчитеся створювати адаптивні зображення за допомогою тега <picture>. Практичні навички роботи з такими інструментами, як CodePen і CodeSandbox, дозволять вам реалізувати веб-проєкти та перевірити взаємодію між локальними та хмарними середовищами.

**Вивчити питання**

1. Загальні відомості:
2. Опис методу:
3. Параметри:
4. Область дії імен:
5. Рекурсивні методи:
6. Перевантаження методів:
7. Методи з необов'язковими параметрами:
8. Іменовані параметри:

***Завдання до роботи:***

1. Виконати домашні завдання описані у теоретичних матеріалах до лекцій 4,5,7

2. Щодо лекції 8 (ООП в JS) завдання наступне:  
- створити щонайменше 3 функції конструктори (класи), Можна застосовувати, навіть бажано, ES6 синтаксис

- проілюструвати на прикладі трьох і більше класах прототипне наслідування в JS

3. Як ви знаєте, викликавши метод **toString** на будь-якому обєкті, наприклад ({}).toString() отримаємо "[object Object]".  ⚠️Увага⚠️, чорна скринька в студію:  
- опишіть як ми можемо інтерпретувати (пояснити) значення вищезгаданого результату "[object Object]"  
- чому [].toString() не виводить "[object Array]"

- реалізувати вивід "[object Array]" для масиву

4. Оформити звіт за взірцем, де представити результати код, скріншоти виконання, пояснення коду чи вирішення проблеми (якщо таке потрібно)

Хід роботи

Користувався <https://www.programiz.com/javascript/online-compiler/>

Онлайн компілятор по js  
Безкоштовний,зручний,приємний інтерфейс.

2. Щодо лекції 8 (ООП в JS) завдання наступне:  
- створити щонайменше 3 функції конструктори (класи), Можна застосовувати, навіть бажано, ES6 синтаксис

- проілюструвати на прикладі трьох і більше класах прототипне наслідування в JS

// Клас Person

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

return `Hello, my name is ${this.name} and I am ${this.age} years old.`;

}

}

// Клас Student, що наслідує від Person

class Student extends Person {

constructor(name, age, studentId) {

super(name, age);

this.studentId = studentId;

}

study() {

return `${this.name} is studying.`;

}

}

// Клас Teacher, що наслідує від Person

class Teacher extends Person {

constructor(name, age, subject) {

super(name, age);

this.subject = subject;

}

teach() {

return `${this.name} teaches ${this.subject}.`;

}

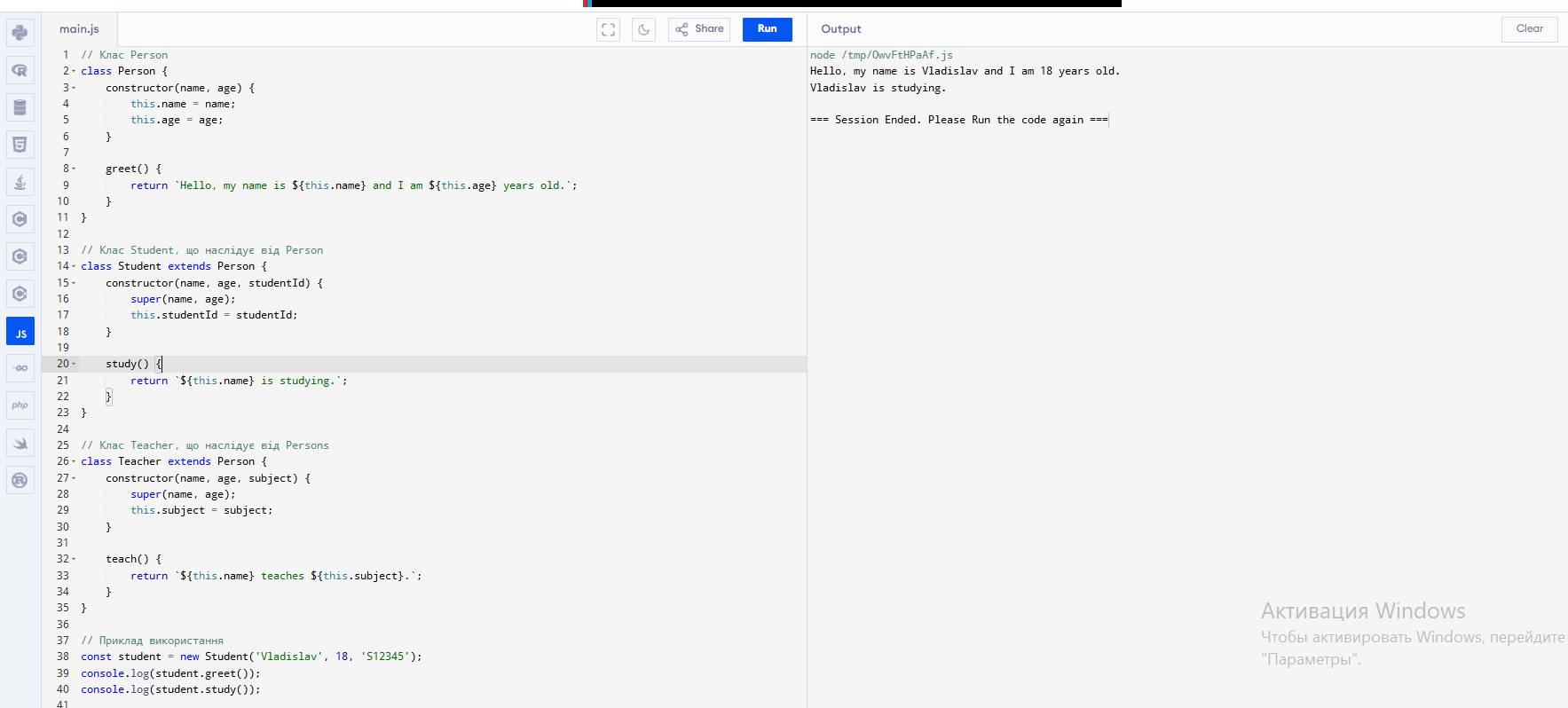
}

// Приклад використання

const student = new Student('Vladislav', 18, 'S12345');

console.log(student.greet());

console.log(student.study());



Коли викликається метод toString на об'єкті, результат у вигляді "[object Object]" слугує для ідентифікації типу об'єкта. Це стандартна поведінка, оскільки метод toString в класичному JavaScript (ECMAScript) визначений в прототипі об'єкта. Тобто, всі об'єкти, які не мають власного методу toString, успадковують його реалізацію, яка повертає рядок у форматі "[object Type]".

Я зазначив, що метод toString для масивів обробляє їх як об'єкти і об'єднує елементи в рядок, відділяючи їх комами. Якщо масив порожній, результат буде просто пустим рядком, оскільки немає елементів для об'єднання. Це важливий момент, оскільки він демонструє, як JavaScript обробляє різні типи даних.

Я показав, як перевизначити метод toString для масивів, додавши кастомну реалізацію, яка повертає "[object Array]". Це дозволяє змінити стандартну поведінку методу для всіх масивів, але я також зазначив, що таке перевизначення слід робити обережно, оскільки воно може призвести до конфліктів з іншими частинами коду.

3. Як ви знаєте, викликавши метод **toString** на будь-якому обєкті, наприклад ({}).toString() отримаємо "[object Object]".  ⚠️Увага⚠️, чорна скринька в студію:  
- опишіть як ми можемо інтерпретувати (пояснити) значення вищезгаданого результату "[object Object]"  
- чому [].toString() не виводить "[object Array]"

- реалізувати вивід "[object Array]" для масиву

// Перевизначення методу toString для масиву

Array.prototype.toString = function() {

return "[object Array]";

};

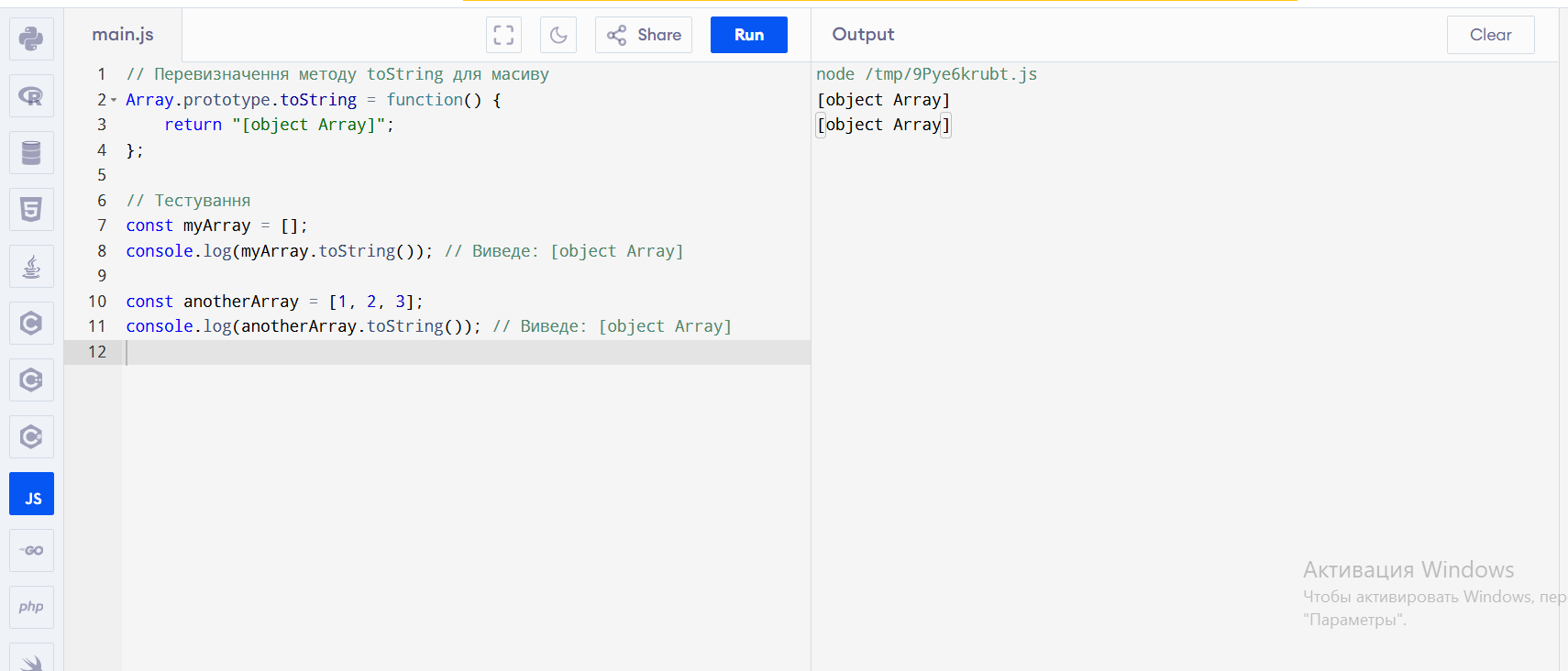
// Тестування

const myArray = [];

console.log(myArray.toString()); // Виведе: [object Array]

const anotherArray = [1, 2, 3];

console.log(anotherArray.toString()); // Виведе: [object Array]



**Висновок:**

У цій лабораторній роботі були освоєні основні методи створення та відправки веб-форм з використанням HTML, CSS та JavaScript, а також розглянуті запити GET і POST. Вивчення різних типів елементів форм дало можливість краще зрозуміти, як користувачі взаємодіють з веб-додатками. Використання тега <picture> для створення адаптивних зображень допомогло вивчити принципи сучасної веб-розробки.

У завданні щодо об'єктно-орієнтованого програмування (ООП) в JavaScript були створені три класи: Person, Student і Teacher. Клас Student успадковує властивості та методи від класу Person, а клас Teacher також наслідує від Person, що ілюструє принцип прототипного наслідування. Завдяки цьому ми зможемо реалізувати більш складну логіку в майбутньому.

Проблема з методом toString була проаналізована, щоб зрозуміти, чому виклик {}.toString() повертає [object Object], тоді як [].toString() не виводить [object Array]. Було продемонстровано, як перевизначити метод toString для масиву, щоб отримати очікуване значення [object Array]. Це відкриває можливість кастомізації стандартних поведінок об'єктів JavaScript, хоча слід дотримуватися обережності, щоб уникнути конфліктів у коді.