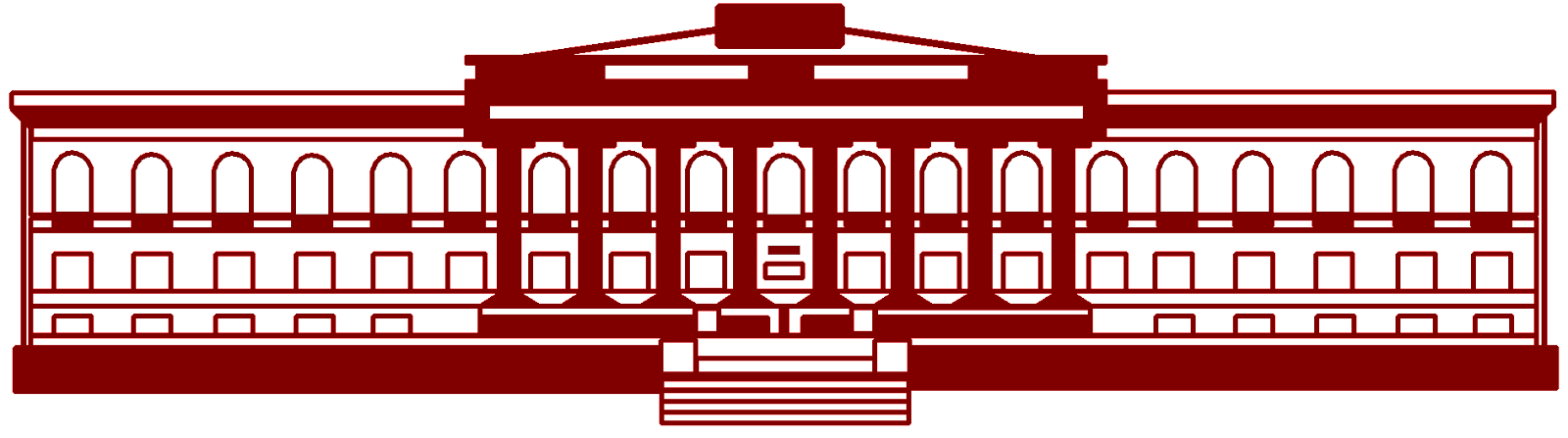
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій

**Кафедра прикладних інформаційних систем**



**Звіт**

до виконання лабораторної роботи №3

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

на тему:

« Знайомство з JavaScript: цикли, типи данних, базова взаємодія з

користувачем »

**Виконано:**

студ. групи ПП- 31, підгрупа 2

Шкандюк Анною Леонідівною

**Перевірено:**

Пирог М.В.

**Київ – 2023**

1. **Мета роботи:**

Отримати навички роботи з основними типами данних, а також створення основних функцій при роботі з кодом: сортування, фільтрація, валідація, пошук.

1. **Завдання та їх виконання:**

**Завдання 1**. Данні з обєкту random-user-mock привести до вигляду:

{

*"gender"*: "male",

*"title"*: "Mr",

*"full\_name"*: "Norbert Weishaupt",

*"city"*: "Rhön-Grabfeld",

*"state"*: "Mecklenburg-Vorpommern",

*"country"*: "Germany",

*"postcode"*: 52640,

*"coordinates"*: { *"latitude"*: "-42.1817", *"longitude"*: "-152.1685" },

*"timezone"*: { *"offset"*: "+9:30", *"description"*: "Adelaide, Darwin" },

*"email"*: "norbert.weishaupt@example.com",

*"b\_date"*: "1956-12-23T19:09:19.602Z",

*"age"*: 65,

*"phone"*: "0079-8291509",

*"picture\_large"*: "https://randomuser.me/api/portraits/men/28.jpg",

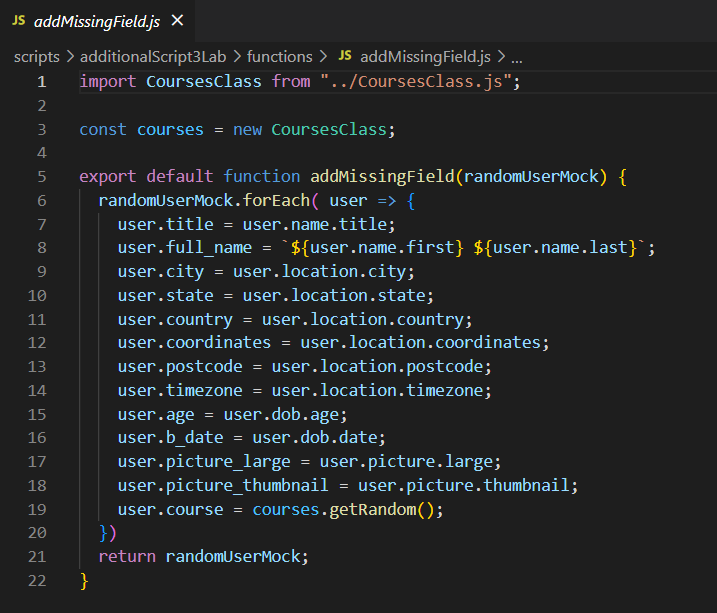
*"picture\_thumbnail"*: "https://randomuser.me/api/portraits/thumb/men/28.jpg"

}

До кожного з об’єктів масиву додати поля: id, favorite, course, bg\_color, note, заповнюючи їх ПРАВИЛЬНИМ типом данних. По’єднати два обєкти (random\_user\_mock та additional\_users) в один, позбуваючись повторів, якщо такі є. Значення поля course заповнювати рандомно зі списку:

Mathematics, Physics, English, Computer Science, Dancing, Chess, Biology, Chemistry, Law, Art, Medicine, Statistics

Обєкти можуть мати не однакову кількість полів та різні інтерфейси. Результатом виконання, є функція, що повертає масив відформатованних об’єктів.

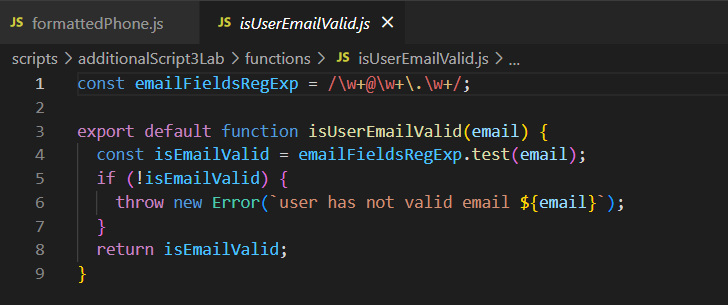
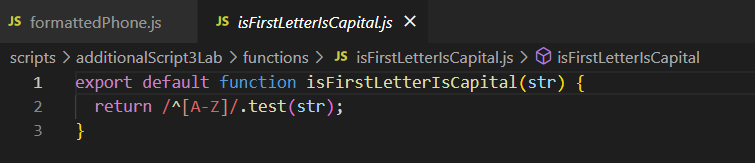
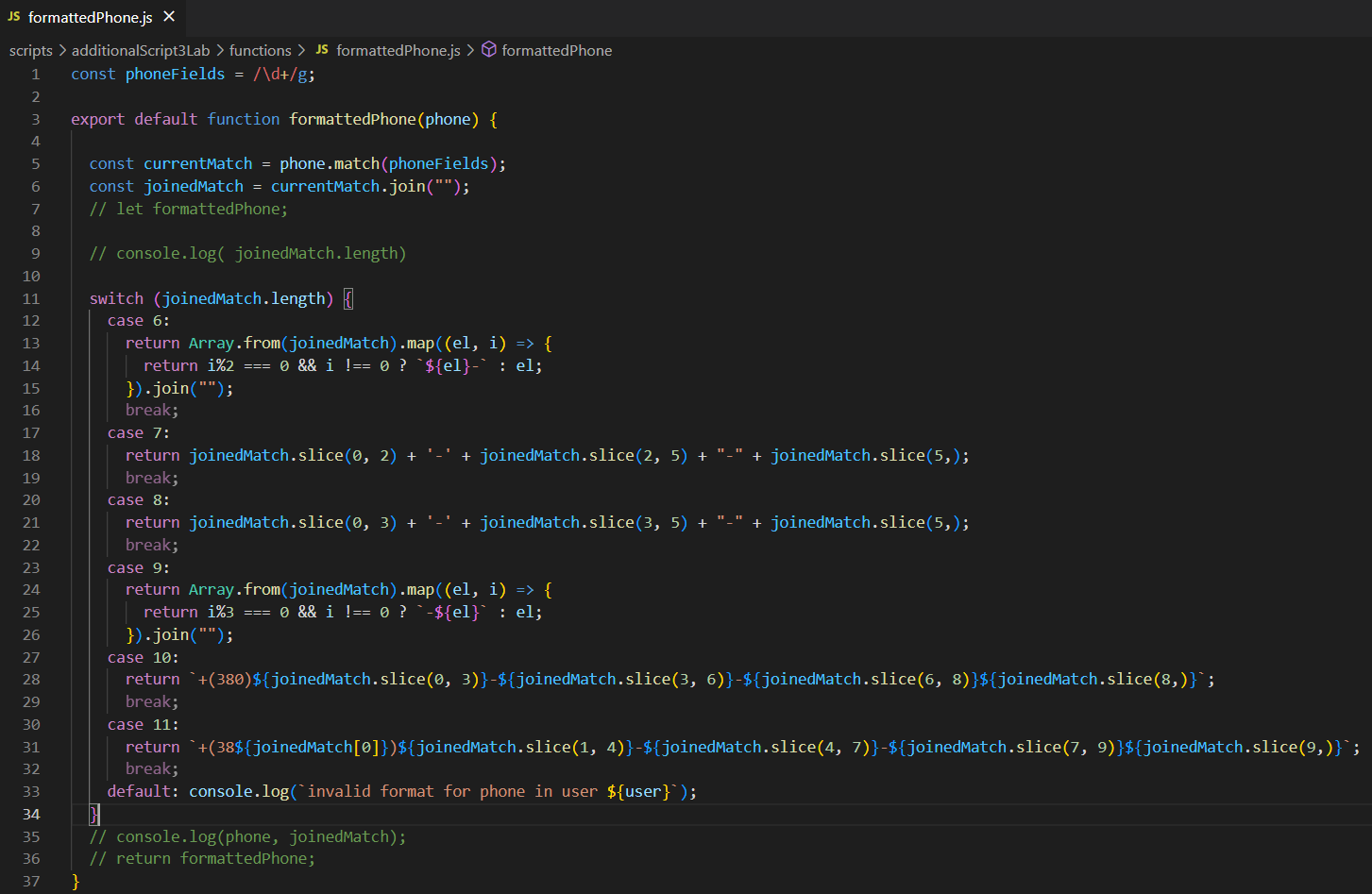


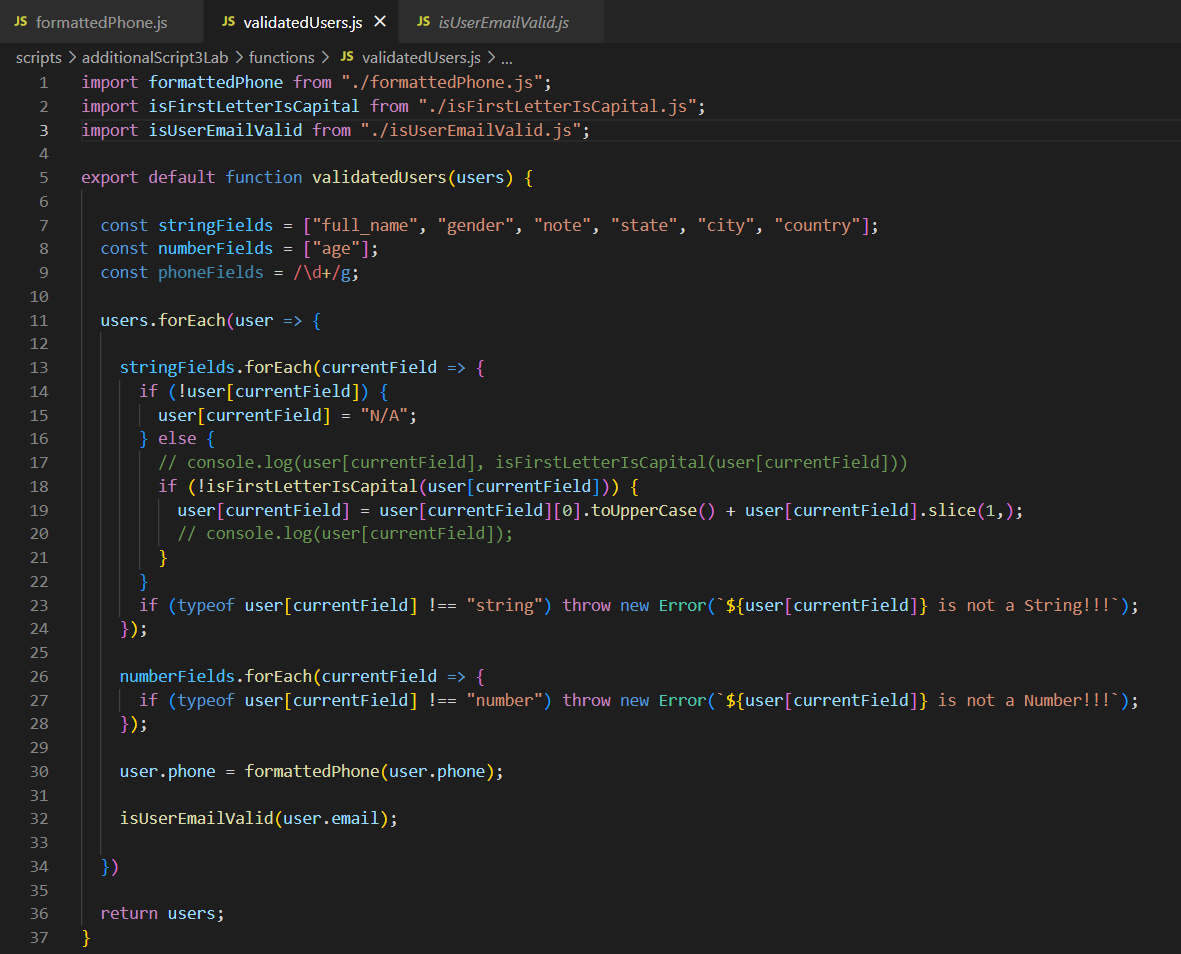
**Завдання 2**. Провалідувати обєкт. Тобто перевірити, чи відповідають поля заданого обєкту вимогам до нього. Валідними вважаються такі поля, які відповідають наступним вимогам: • Поля full\_name, gender, note, state, city, country мають бути строками, та починатись з великої літери.

• Поле age має бути чисельним.

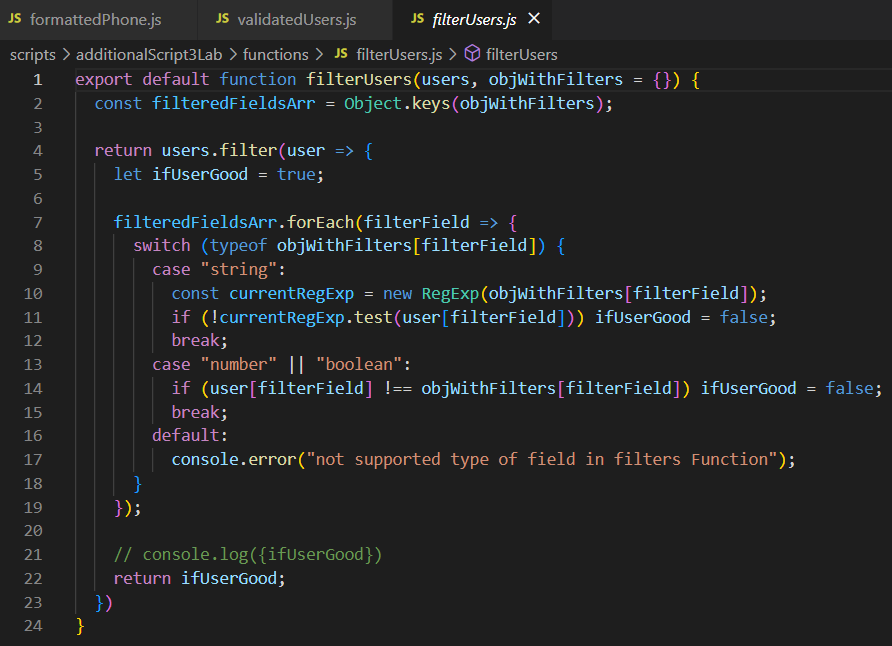
• Поле phone має відповідати заданому формату (формат залежить від країни).

• Поле email має відповідати формату запису email, тобто мати @.

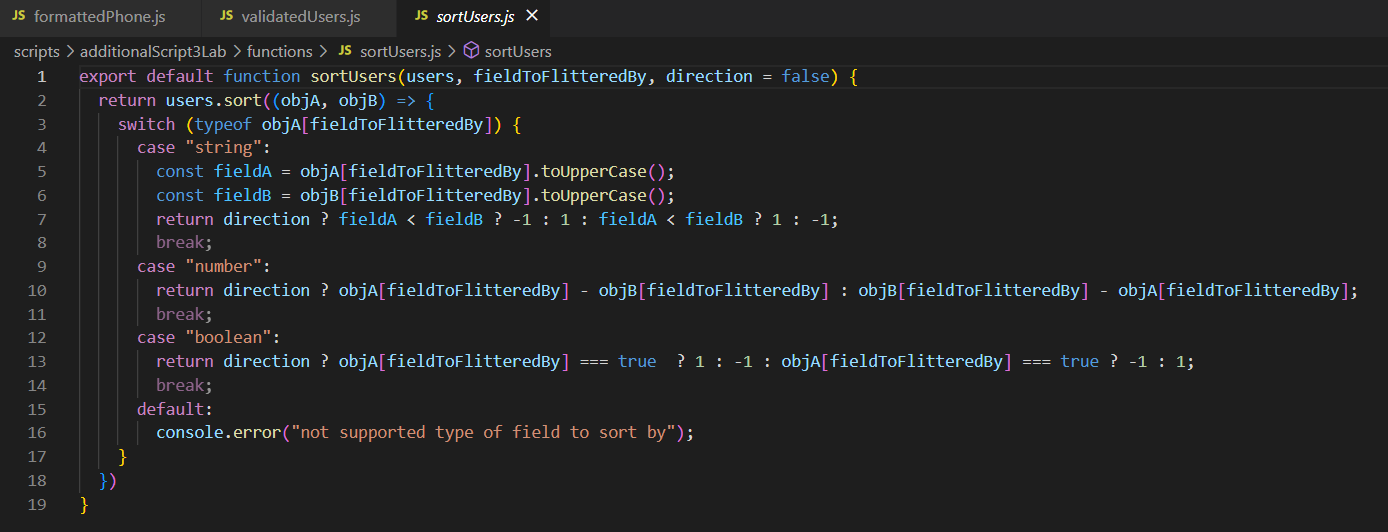




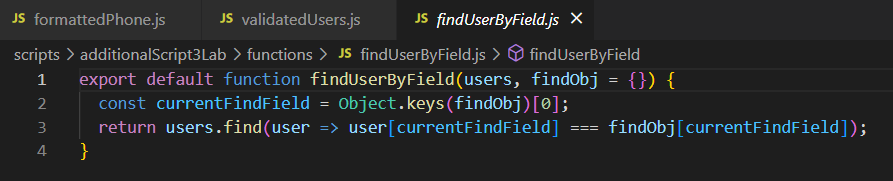
**Завдання 3**. Написати функцію фільтрації массиву обєктів за параметрами (параметри змінними). Параметри є полями обєкту: country, age, gender, favorite. Фільтрація повинна працювати як логічне «і».



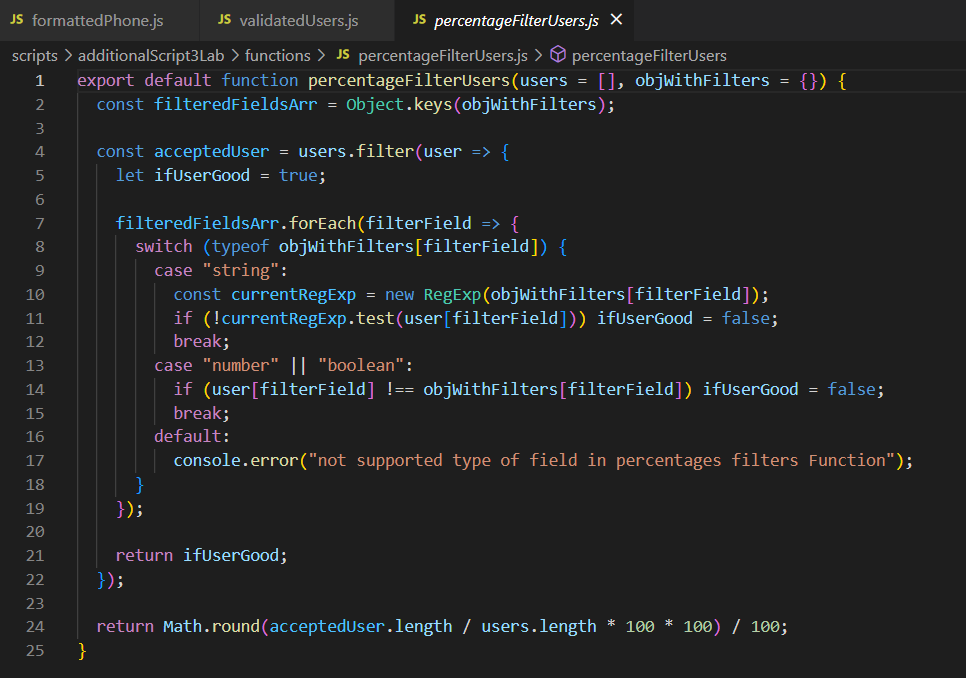
**Завдання 4.** Написати функцію сортування массиву обєктів за параметрами (параметри змінними). Сортування може бути як за зростанням так і за спаданням. Сортуватись можуть чисельні поля та строкові: full\_name, age, b\_day, country. Сортування працює по одному парамету (логічне «або»)



**Завдання 5.** Знайти в массиві об’єкт, який відповідає параметру пошуку. Параметром може бути як строкове, так і чисельне поле: name, note, age.



**Завдання 6.** Написати функцію, яка повертає відсоток від загального числа обєктів в массиві, що відповідають пошуку. Тобто, якщо у нас пошук за віком більше 30, то функція поверне число відсотків, які відповідають кількості юзерів. Пр: загальна кількість юзерів – 50, з них 30 за віком більше 30 років, то функція поверне 60.



1. **Висновки до роботи**

За результатами проробленої роботи можемо сказати, що отримали навички роботи з основними типами данних, а також створили основні функцій при роботі з кодом: сортування, фільтрація, валідація, пошук.

1. **Контрольні запитання**
2. Як можна підключити js-код на сторінку?

Є декілька способів підключити JavaScript-код на веб-сторінку

1. Вставка прямо в HTML:

Ви можете вставити JavaScript-код безпосередньо всередину тегу `<script>` у вашому HTML-документі. Рекомендовано розміщувати цей код в кінці тегу `<body>`, перед закриваючим `</body>` тегом, для оптимальної продуктивності.

<!DOCTYPE html>

<html>

<head>

<title>Моя сторінка</title>

</head>

<body>

<!-- Ваш вміст тут -->

<script>

// Ваш JavaScript-код тут

</script>

</body>

</html>

2. Зовнішній файл:

Рекомендовано розміщувати ваш JavaScript-код у зовнішньому файлі з розширенням `.js`. Для цього створіть окремий файл, наприклад, `script.js`, і підключіть його до вашої HTML-сторінки за допомогою тегу `<script>` з атрибутом `src`.

Вміст `script.js`:

// Ваш JavaScript-код тут

HTML:

<!DOCTYPE html>

<html>

<head>

<title>Моя сторінка</title>

<script src="script.js"></script>

</head>

<body>

<!-- Ваш вміст тут -->

</body>

</html>

3. Асинхронне або відкладене завантаження:

Якщо вам потрібно завантажити JavaScript-код асинхронно або відкладено, ви можете використовувати атрибути `async` або `defer` у тегу `<script>`.

- `<script async src="script.js"></script>`: Код завантажується асинхронно, не зупиняючи обробку HTML-сторінки.

- `<script defer src="script.js"></script>`: Код завантажується асинхронно, але виконується тільки після завантаження всього HTML-документа.

1. Які бувають оператори в JS (хоча б 5)?

JavaScript має багато різних операторів, які використовуються для виконання різних дій над значеннями. Ось п'ять базових операторів:

1. Арифметичні оператори:

- `+`: Додавання

- `-`: Віднімання

- `\*`: Множення

- `/`: Ділення

- `%`: Остача від ділення

2. Оператори присвоєння:

- `=`: Присвоєння значення

- `+=`: Додавання та присвоєння

- `-=`: Віднімання та присвоєння

- `\*=`: Множення та присвоєння

- `/=`: Ділення та присвоєння

3. Оператори порівняння:

- `==`: Рівність

- `!=`: Нерівність

- `===`: Рівність по значенню та типу (строге порівняння)

- `!==`: Нерівність по значенню та типу

- `>`: Більше

- `<`: Менше

- `>=`: Більше або рівно

- `<=`: Менше або рівно

4. Логічні оператори:

- `&&`: Логічне "І" (AND)

- `||`: Логічне "АБО" (OR)

- `!`: Логічне "НЕ" (NOT)

5. Оператори інкремента/декремента:

- `++`: Інкремент (збільшення на одиницю)

- `--`: Декремент (зменшення на одиницю)

1. У чому відмінність між JavaScript та ECMAScript?

JavaScript та ECMAScript (ES) - це терміни, які часто використовуються в контексті програмування на мові JavaScript. Ось їх відмінності:

1*. JavaScript*:

- JavaScript - це скриптова мова програмування, яка використовується для створення динамічних веб-сторінок.

- Це мова, яку розуміють браузери та яка використовується для взаємодії зі сторінкою, маніпулювання DOM, валідації форм та багатьох інших завдань на клієнтському боці.

- JavaScript може включати розширення та функції, які не входять до стандарту ECMAScript.

2. *ECMAScript*:

- ECMAScript - це стандарт, на якому базується мова JavaScript. Він визначає синтаксис та функціональність мови.

- ECMAScript почав розвиватися з метою створення уніфікованого стандарту для скриптової мови, яка би працювала в різних браузерах та середовищах.

- JavaScript є лише однією з реалізацій стандарту ECMAScript. Існують інші реалізації, такі як JScript від Microsoft.

Відмінність між ними полягає в тому, що JavaScript - це конкретна мова програмування, а ECMAScript - це стандарт, який визначає основні правила та функціональність цієї мови. Коли ми говоримо про нові версії JavaScript, ми в основному вказуємо на нові версії стандарту ECMAScript (наприклад, ES6, ES7 тощо), які включають нові функції, синтаксис та поліпшення.

1. У чому відмінність for … in ... та for … of …?

`for...in` та `for...of` - це дві різні структури циклів у JavaScript, і вони мають різні використання та властивості.

`for...in`:

Цей цикл використовується для ітерації (перебору) ключів (імен властивостей) об'єкта. Він призначений для перебору властивостей об'єкта, а не для перебору значень масивів. Властивості об'єкта включають не лише власні властивості, але й властивості, успадковані з прототипу.

for (let key in object) {

// код для властивості key

}

`for...of`:

Цей цикл використовується для ітерації (перебору) значень масивів, рядків, об'єктів, що втілюють ітератор, і деяких інших колекцій. Він призначений для перебору значень, а не ключів.

for (let element of array) {

// код для елементу

}

Отже, основна відмінність між ними полягає в тому, що `for...in` використовується для перебору властивостей об'єкта за їхніми ключами, а `for...of` використовується для перебору значень колекцій, таких як масиви. Також важливо пам'ятати, що `for...of` працює тільки з об'єктами, які мають властивість-ітератор, що дозволяє ітерувати їхні значення.

1. Яка відмінність між while() do{} та do{} while()?

`while` та `do...while` - це дві різні структури циклів у JavaScript, які використовуються для повторення блоку коду доти, доки задана умова вірна. Основна відмінність між ними полягає в тому, коли умова перевіряється.

`while` цикл:

У цьому циклі спочатку перевіряється умова, і якщо вона є вірною, то блок коду виконується.

while (умова) {

// код

}

Примітка: Якщо умова вже на початку не виконується, блок коду може ніколи не виконатися.

`do...while` цикл:

У цьому циклі спочатку виконується блок коду, а потім перевіряється умова. Це означає, що блок коду виконується щонайменше один раз, навіть якщо умова невірна.

do {

// код

} while (умова);

Після виконання блоку коду у `do...while` циклі, умова перевіряється. Якщо вона вірна, цикл продовжується і виконується ще раз. Якщо умова невірна, цикл завершується.

Зауважте, що `do...while` зазвичай використовується, коли ви хочете, щоб блок коду виконався принаймні один раз, незалежно від умови. У випадку `while` цикла, блок коду може не виконатися зовсім, якщо умова на початку вже є невірною.

1. Function expression vs function declaration

`Function Expression` та `Function Declaration` - це два способи оголошення функцій у JavaScript. Ці способи мають різницю у тому, як функції піднімаються (hoisting) та доступні в коді.

Function Declaration (Оголошення функції):

Ви можете оголосити функцію за допомогою ключового слова `function`. Оголошення функції піднімається (hoisted), що означає, що ви можете викликати функцію перед її фактичним оголошенням у коді. Функції, оголошені таким чином, доступні на всій ділянці коду, включаючи перед їхнім оголошенням.

foo(); // Можна викликати перед оголошенням

function foo() {

console.log("Hello from foo!");

}

Function Expression (Вираз функції):

Функцію можна також оголосити за допомогою виразу. У цьому випадку, функція визначається та присвоюється змінній. Функціональний вираз не піднімається (hoisted), тому ви не можете викликати функцію перед її оголошенням у коді. Вона буде доступна тільки після того, як ви присвоїли функціональний вираз змінній.

// foo(); // Виклик видасть помилку, оскільки функція ще не оголошена

const foo = function() {

console.log("Hello from foo!");

};

foo(); // Можна викликати після оголошення

Оголошення функцій допомагають структурувати код та робити його більш зрозумілим. Вибір між `Function Declaration` та `Function Expression` залежить від використання та практик програмування.

1. Function context

Контекст функції (Function Context) - це спеціальне поняття в JavaScript, яке вказує на об'єкт, до якого належить поточна функція. Визначення цього контексту впливає на те, як функція отримує доступ до змінних, властивостей та методів.

У JavaScript контекст функції може бути двох типів:

1. *Global Context* (Глобальний контекст):

У глобальному контексті функції функції виконуються поза будь-якими об'єктами. Вони мають доступ до глобальних змінних та можуть змінювати їх значення.

Приклад:

const globalVar = "Global variable";

function showGlobalVar() {

console.log(globalVar);

}

showGlobalVar(); // Виведе "Global variable"

2. *Object Context* (Контекст об'єкта):

Коли функція викликається в контексті об'єкта, вона називається методом об'єкта. Функція отримує доступ до властивостей та методів цього об'єкта через ключове слово `this`, яке вказує на поточний об'єкт.

Приклад:

const person = {

firstName: "John",

lastName: "Doe",

fullName: function() {

return this.firstName + " " + this.lastName;

}

};

console.log(person.fullName()); // Виведе "John Doe"

У JavaScript контекст функції може змінюватися залежно від того, як викликається функція. Це може призводити до певних непорозумінь та проблем, особливо при вкладених функціях. Для визначення поточного контексту часто використовується ключове слово `this`. Однак слід пам'ятати, що використання `this` в різних ситуаціях може призводити до різних результатів, тому ретельне розуміння контексту функцій є важливим аспектом програмування на JavaScript.

1. Об’єкти, копіювання об’єктів, властивості об’єктів

*Об'єкти в JavaScript*:

Об'єкти - це одна з основних структур даних в JavaScript, яка дозволяє зберігати дані у вигляді пар "ключ-значення". Вони можуть містити різні типи даних, включаючи інші об'єкти, масиви, функції тощо.

const person = {

firstName: "John",

lastName: "Doe",

age: 30,

isStudent: false

};

*Копіювання об'єктів*:

Копіювання об'єктів може бути трохи заплутаною задачею через те, що об'єкти є посиланнями в JavaScript. Коли ви копіюєте об'єкт, ви фактично копіюєте посилання на об'єкт, а не його вміст.

Є кілька способів створення копії об'єкта:

1. Копіювання за допомогою `Object.assign()`:

const copy = Object.assign({}, person);

2. Копіювання за допомогою оператора розширення (`...`):

const copy = { ...person };

3. Глибоке копіювання за допомогою бібліотек (наприклад, `Lodash`):

const deepCopy = \_.cloneDeep(person);

*Властивості об'єктів*:

Властивості об'єкта - це пари "ключ-значення", де ключ - це ім'я властивості, а значення - це дані, пов'язані з цією властивістю.

console.log(person.firstName); // Виведе "John"

console.log(person.age); // Виведе 30

Властивості об'єктів можуть містити різні типи даних, включаючи рядки, числа, булеві значення, масиви, об'єкти та інші. Можна звертатися до властивостей об'єктів для отримання або зміни їхніх значень.

1. Методи обробки масиву

У JavaScript масиви мають велику кількість вбудованих методів, які допомагають вам взаємодіяти з масивами, виконувати операції фільтрації, мапування, знаходження елементів тощо. Ось кілька з найбільш поширених методів обробки масиву:

1. `forEach(callback)`: Виконує функцію зворотнього виклику (callback) для кожного елемента масиву.

const numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number) {

console.log(number);

});

2. `map(callback)`: Створює новий масив, у якому кожен елемент є результатом виклику функції зворотнього виклику для відповідного елемента в початковому масиві.

const doubledNumbers = numbers.map(function(number) {

return number \* 2;

});

3. `filter(callback)`: Створює новий масив, що містить лише елементи, для яких функція зворотнього виклику повертає `true`.

const evenNumbers = numbers.filter(function(number) {

return number % 2 === 0;

});

4. `find(callback)`: Повертає перший елемент, для якого функція зворотнього виклику повертає `true`.

const firstEvenNumber = numbers.find(function(number) {

return number % 2 === 0;

});

5. `reduce(callback, initialValue)`: Застосовує функцію зворотнього виклику до всіх елементів масиву, збираючи їх в одне значення (наприклад, сума або добуток).

const sum = numbers.reduce(function(acc, current) {

return acc + current;

}, 0); // 0 - початкове значення аккумулятора

1. Прототипи та успадкування

Прототипи та успадкування - це важливі концепції в об'єктно-орієнтованому програмуванні, включаючи JavaScript.

Прототипи:

У JavaScript кожен об'єкт має асоційований об'єкт-прототип, який визначає базові властивості та методи, що спільні для всіх об'єктів. Коли звертаєтеся до властивості або методу об'єкта, який не має такої властивості або методу, JavaScript буде шукати їх у його прототипі, а також у ланцюжку прототипів (прототипів прототипів) до тих пір, поки не знайде властивість чи метод або не дійде до кінця ланцюжка.

Успадкування:

Успадкування дозволяє створювати нові об'єкти на основі існуючих об'єктів, додаючи до них нові властивості та методи або змінюючи вже існуючі. Це допомагає створювати ієрархію класів, де підкласи успадковують властивості та методи від своїх батьківських класів.

В JavaScript успадкування реалізується за допомогою прототипів. Коли ви створюєте новий об'єкт або функцію-конструктор, ви можете вказати, який об'єкт-прототип буде використовуватися для успадковування.

Приклад успадкування за допомогою прототипів:

// Батьківський об'єкт (прототип)

const Animal = {

makeSound: function() {

console.log("Animal makes a sound");

}

};

// Створення нового об'єкта на основі прототипу Animal

const cat = Object.create(Animal);

cat.makeSound(); // "Animal makes a sound"

// Додавання додаткових властивостей та методів до об'єкта cat

cat.name = "Whiskers";

cat.meow = function() {

console.log("Meow!");

};

cat.meow(); // "Meow!"

Успадкування за допомогою прототипів дозволяє реалізувати концепцію розділення коду та реюзабельності, що є ключовими принципами об'єктно-орієнтованого програмування.