

Колекції. Словники. Множини

Map – це тип даних:

- є колекцією типу **ключ/значення** (за аналогією до об’єктів Object, але є ряд відмінностей)
- спільним у Map і Object є те, що обидва типи даних є колекціями типу **ключ/значення**
- основною відмінністю між Map і Object є те, що Map як ключі може використовувати не тільки String, але і величини інших типів. У звичайних об’єктів якщо при описі ключ не String, то він автоматично приводиться до String.

**Створення об’єктів Map**

Метод	Загальна форма	Приклад
Створення порожньої колекції	<b>new Map()</b>	let map = new <b>Map</b> ()
Створення з масиву пар [ключ, значення]  (ключі можуть бути довільного типу)	<b>new Map</b> ( [ [ключ1, значення 1], [ключ2, значення 2], ..... ] )	let map = new <b>Map</b> ( [ [ '1', 'str1'], [ 1, 'num1'], [ true, 'bool1'] ] )  // Map(3) {'1' => 'str1', 1 => 'num1', true => 'bool1'}
Створення з об’єкта з використанням <b>Object.entries(obj)</b> (властивості і значення буде додано до map)	<b>new Map</b> ( Object.entries( <b>об’єкт</b> ) )	let obj = { name: "Іван", age: 30 };  let map = new Map(Object.entries(obj))  alert( map.get('name') ); // Іван

Методи Map

Метод	Загальна форма	Приклад
map.set(key, value)	Додавання нової пари ключ/значення	map.set('1', 'str1') // рядок як ключ map.set(1, 'num1') // цифра як ключ map.set(true, 'bool1') // булеве значення як ключ
map.get(key)	Отримання значення за ключем	alert( map.get(1) ); // 'num1' alert( map.get('1') ); // 'str1'
map.has(key)	повертає true якщо key існує, інакше false	alert( map.has(1) ) //true alert( map.has(77) ) //false
map.size	повертає поточну кількість елементів	alert( map.size ) // 3
map.delete(key)	видаляє елемент по ключу	map.delete(key) alert( map.size ) // 2 (залишились пари з ключами '1', true)
map.clear()	видаляє всі елементи колекції	map.clear() alert( map.size ) // 0

## Перегляд елементів Map

Метод	Загальна форма	Приклад для тестової колекції <code>let map = new Map( [ [ '1', 'str1'], [1, 'num1'], [true, 'bool1'] ] )</code>
З використанням циклу <b>for..of</b>	<code>for (let entry of map) {     . . . . }</code>	<code>for (let entry of map) {     alert(entry)      // 1,str1 // 1,num1 // true,bool1 }</code>
З використанням <b>forEach</b>	<code>map.<i>forEach</i>(     (value, key, map) =&gt; {...} )</code>	<code>map.<i>forEach</i>( (value, key, map) =&gt; {     alert(`\${key}: \${value}`); }); // 1:str1 // 1:num1 // true:bool1</code>

Властивості Map

Метод	Загальна форма	Приклад для тестової колекції let map = new Map( [ [ '1', 'str1'], [1, 'num1'], [true, 'bool1'] ] )
Ітератор ключів map.keys()	повертає об'єкт-ітератор для ключів	for (let key of map.keys()) { alert(key)      // '1', 1, true } //----- Array.from(map.keys()) // ['1', 1, true ]
Ітератор значень map.values()	повертає об'єкт-ітератор для значень	for (let value of map.values()) { alert(value)     // 'str1', 'num1', 'bool1' } //----- Array.from(map.keys()) // ['str1', 'num1', 'bool1']
Ітератор пар [ключ, значення] map.entries()	повертає об'єкт-ітератор зі значеннями виду [ключ, значення], цей варіант використовується за умовчанням у for..of	for (let entry of map) { alert(entry)     // 1,str1 // 1,num1 // true,bool1}



Приклад. Дано список з віком учнів. Підрахувати скільки разів кожне значення зустрічається у списку і максимум

Приклад. Дано масив книг (назва, рік видання, автор (ПІБ, рік народження)). Підрахувати кількість книг для кожного автора

# Концептуальна основа Map - Хеш-таблиця

Map у JavaScript реалізований як хеш-таблиця (hash table), яка є ефективною структурою даних для асоціативного зберігання пар "ключ-значення".

Основні компоненти хеш-таблиці в Map:

## 1.Масив бакетів (buckets):

- 1. Хеш-таблиця містить масив фіксованого розміру, де кожен елемент (бакет) — це список або інша структура для зберігання пар "ключ-значення".
- 2. Розмір масиву зазвичай є степенем двійки (наприклад, 16, 32, 64), щоб оптимізувати хешування.

## 2.Хеш-функція:

- 1. Кожен ключ (будь-якого типу: рядок, число, об'єкт, символ тощо) пропускається через хеш-функцію, яка перетворює його в числовий індекс для масиву бакетів.
- 2. Хеш-функція має бути:
  - 1. Детермінованою: той самий ключ завжди дає той самий хеш.
  - 2. Рівномірною: хеші розподіляються по бакетах максимально рівно, щоб мінімізувати колізії.

## 3.Вирішення колізій:

- 1. Колізії виникають, коли два різні ключі хешуються в той самий індекс. Для їх вирішення використовуються:
  - 1. Окреме зв'язування (separate chaining): кожен бакет містить зв'язаний список або масив пар "ключ-значення".
  - 2. Відкрита адресація (open addressing): пошук іншого вільного бакета за допомогою пробування.
- 2. У сучасних рушіях, таких як V8, для зменшення накладних витрат при багатьох колізіях можуть використовуватися дерева (наприклад, AVL або червоно-чорні дерева) замість списків.

## 4.Пари ключ-значення:

- 1. Кожна пара зберігається як запис, що містить ключ, значення та, можливо, додаткові метадані (наприклад, посилання для порядку вставки).

# Приклад, як можна би реалізувати самотійно

```
class SimpleMap {
  constructor() {
    this.buckets = new Array(4); // 4 бакети
    this.size = 0;
    this.order = []; // Для збереження порядку вставки
  }
  // Хеш-функція: довжина рядка % кількість бакетів
  hash(key) {
    const str = String(key);
    return str.length % this.buckets.length;
  }
  // Додавання пари ключ-значення
  set(key, value) {
    const index = this.hash(key);
    if (!this.buckets[index]) {
      this.buckets[index] = [];
    }
    const bucket = this.buckets[index];
    const existing = bucket.find(item => item[0] === key);
    if (existing) { existing[1] = value; // Оновлення } else {
      bucket.push([key, value]); // Додавання нової пари
      this.order.push(key); // Зберігаємо порядок
      this.size++;
    }
  }
  // Отримання значення
  get(key) {
    const index = this.hash(key);
    const bucket = this.buckets[index];
    if (!bucket) return undefined;
    const item = bucket.find(item => item[0] === key);
    return item ? item[1] : undefined;
  }
}
```

```
// Використання
const map = new SimpleMap();

// Додаємо ключі, які викликають колізію
map.set("cat", "meow"); // Довжина "cat" = 3 → hash = 3 % 4 = 3
map.set("dog", "woof"); // Довжина "dog" = 3 → hash = 3 % 4 = 3 (колізія!)
map.set("bird", "tweet"); // Довжина "bird" = 4 → hash = 4 % 4 = 0
map.set("fish", "bubble"); // Довжина "fish" = 4 → hash = 4 % 4 = 0 (колізія!)
```

```
+-----+
| buckets (масив із 4 бакетів) |
+-----+
| Index 0 | -> [ ["bird", "tweet"] |
|          | ["fish", "bubble"] ] |
+-----+
| Index 1 | -> empty |
+-----+
| Index 2 | -> empty |
+-----+
| Index 3 | -> [ ["cat", "meow"] |
|          | ["dog", "woof"] ] |
+-----+
```



## Множини. Set

Об’єкт Set :

- це особливий тип колекції: “множина” значень (без ключів)
- кожне значення може з’являтися тільки раз (тобто множина значень є унікальною).

### Створення об’єктів Set

Метод	Загальна форма	Приклад
Створення порожньої колекції	<code>new Set()</code>	<code>let set = new Set()</code>
Створення з ітерованого об’єкта	<code>new Set (</code> <div><code>ітерований_об’єкт</code></div> <code>)</code>	<code>let set = new Set(["апельсини", "яблука", "банани"]);</code>  <code>for (let value of set) alert(value); // 'апельсини', 'яблука', 'банани'</code>  <code>//Set(3) {'апельсини', 'яблука', 'банани'}</code>

Методи Set

Метод	Загальна форма	Приклад
set. add (value)	Додавання нового значення	let set = new Set();  let ivan = { name: "Іван" }; let petro = { name: "Петро" }; let maria = { name: "Марія" };  set.add(ivan); set.add(petro); set.add(maria); set.add(ivan); set.add(maria);
set. has (value)	повертає true, якщо value присутнє в множині Set, інакше false	set. has (ivan) //true
set.size	повертає поточну кількість елементів	alert(set.size ) // 3
set.delete(value)	видаляє елемент	set.delete(ivan) alert(set.size ) // 2 (залишились: petro, maria)
set.clear()	видаляє всі елементи колекції	set.clear() alert(set.size ) // 0

## Перегляд елементів. Властивості Set

Метод	Загальна форма	Приклад для тестової колекції <code>let set = new Set(["апельсини", "яблука", "банани"]);</code>
З використанням циклу <code>for..of</code>	<pre>for (let value of set) {   . . . }</pre>	<pre>for (let value of set ()) {   alert(key)    //  "апельсини", "яблука", "банани" } //----- Array.from(set.keys()) // ["апельсини", "яблука", "банани"]</pre>
З використанням  <code>forEach</code>	<pre>set.forEach(   (value, valueAgain, set) =&gt; {...} )  (value == valueAgain)</pre>	<pre>set.forEach((value, valueAgain, set) =&gt; {   alert(value);  "апельсини", "яблука", "банани" });</pre>

## Властивості Set

Метод	Загальна форма	Приклад для тестової колекції <code>let set = new Set(["апельсини", "яблука", "банани"]);</code>
Ітератор значень <code>set.values()</code>	повертає об'єкт-ітератор для значень	<pre>for (let value of set.values()) {   alert(value)    // "апельсини", "яблука", "банани" } //----- Array.from(set.keys()) // ["апельсини", "яблука", "банани"]</pre>
Знову ітератор значень (ключів немає) <code>set.keys()</code>	повертає об'єкт-ітератор для значень	<pre>for (let key of set.keys()) {   alert(key)      // "апельсини", "яблука", "банани" } //----- Array.from(set.keys()) // ["апельсини", "яблука", "банани"]</pre>
Ітератор пар [значення, значення] <code>set.entries()</code>	повертає об'єкт-ітератор зі значеннями виду [значення, значення]	<pre>for (let entry of set.entries())   alert(entry) //  апельсини,апельсини // яблука, яблука // банани, банани //----- &gt; Array.from(set.entries()) &lt; ▼ (3) [Array(2), Array(2), Array(2)] ⓘ   ▶ 0: (2) ['апельсини', 'апельсини']   ▶ 1: (2) ['яблука', 'яблука']   ▶ 2: (2) ['банани', 'банани']     length: 3</pre>

Приклад. Дано список з віком учнів. Підраховувати скільки є різних значень.

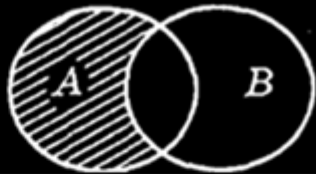

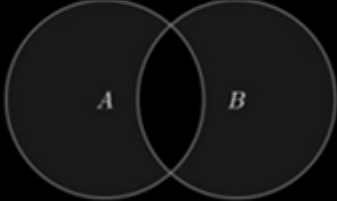
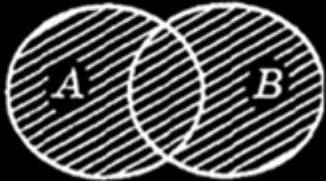
Приклад. Дано масив книг (назва, рік видання, автор (ПІБ, рік народження)).

Підрахувати кількість різних авторів



Операції з множинами

```
let s1 = new Set([1,2,3])
let s2 = new Set([3, 4, 5])
```

Метод	Тип поверненого значення	Математичний еквівалент	Діаграма Венна
<div><div>A.difference(B)</div><div>s1.difference(s2) ► Set(2) {1, 2}</div></div>	<div>Set</div>	$A \setminus B$	<div><div>(A \ B)</div><div>Різниця <math>A \setminus B</math> є множина, що складається в точності з усіх елементів <math>A</math>, які не належать до <math>B</math></div></div>
<div><div>A.intersection(B)</div><div>s1.intersection(s2) ► Set(1) {3}</div></div>	<div>Set</div>	$A \cap B$	<div><div>(A \cap B)</div><div>Перетин (добуток) <math>A \cap B</math> є множина, що містить тільки елементи, які належать до <math>A</math> і <math>B</math> одночасно (рис. 1.8).</div></div>
<div><div>A.symmetricDifference(B)</div><div>s1.symmetricDifference(s2) ► Set(4) {1, 2, 4, 5}</div></div>	<div>Set</div>	$(A \setminus B) \cup (B \setminus A)$	<div><div>Множина, що складається з елементів, множини <math>A</math> або множини <math>B</math>, але не є їх спільними елементами</div></div>
<div><div>A.union(B)</div><div>s1.union(s2) ► Set(5) {1, 2, 3, 4, 5}</div></div>	<div>Set</div>	$A \cup B$	<div><div>(A \cup B)</div><div>Об'єднання (сума) <math>A \cup B</math> є множина, що складається з тих і тільки тих елементів, які входять або до <math>A</math>, або до <math>B</math>, або до <math>A</math> і <math>B</math> одночасно (рис. 1.7).</div></div>
<div>A.isDisjointFrom(B)</div>	<div>Boolean</div>	$A \cap B = \emptyset$	Чи множини не мають спільних елементів?
<div>A.isSubsetOf(B)</div>	<div>Boolean</div>	$A \subseteq B$	Чи усі елементи у $A$ є також у множині $B$ ?
<div>A.isSupersetOf(B)</div>	<div>Boolean</div>	$A \supseteq B$	Чи усі елементи у $B$ є також у множині $A$ ?

### **1)Об'єднання множин (Union):**

- Дано перелік ідентифікаційних кодів клієнтів закладу за два дні (деякі з клієнтів можуть відвідати заклад протягом дня декілька разів). Створити список ідентифікаторів клієнтів, які протягом цих двох днів відвідали заклад принаймні один раз.

### **2)Перетин множин (Intersection):**

- Дано перелік ідентифікаційних кодів клієнтів закладу за два дні (деякі з клієнтів можуть відвідати заклад протягом дня декілька разів). Створити список ідентифікаторів клієнтів, які відвідали заклад і в перший, і в другий день

### **3)Різниця множин (Difference):**

- Дано перелік ідентифікаційних кодів клієнтів закладу за два дні (деякі з клієнтів можуть відвідати заклад протягом дня декілька разів). Створити список ідентифікаторів клієнтів, які відвідали заклад тільки у перший день

### **4)Підмножина (Subset):**

- Дано перелік ідентифікаційних кодів клієнтів закладу за два дні (деякі з клієнтів можуть відвідати заклад протягом дня декілька разів). Перевірити, чи усі клієнти, що відвідали заклад у перший день прийшли також і у другий день.

### **5) Symetric difference :**

- Дано перелік ідентифікаційних кодів клієнтів закладу за два дні (деякі з клієнтів можуть відвідати заклад протягом дня декілька разів). Створити список ідентифікаторів клієнтів, які відвідали заклад тільки одного дня (або у перший день, або у другий день).

## Приклад, як можна би реалізувати самостійно (аналогічно, за допомогою хеш-таблиць)

```
class SimpleSet {
  constructor() {
    this.buckets = new Array(4); // 4 бакети
    this.size = 0;
    this.order = []; // Для збереження порядку вставки
  }
  // Хеш-функція: довжина рядка % кількість бакетів
  hash(value) {
    const str = String(value);
    return str.length % this.buckets.length;
  }
  // Додавання значення
  add(value) {
    const index = this.hash(value);
    if (!this.buckets[index]) { this.buckets[index] = []; }
    const bucket = this.buckets[index];
    if (!bucket.some(item => this.isEqual(item[0], value))) {
      bucket.push([value, true]); // Додаємо значення з маркером
      this.order.push(value); // Зберігаємо порядок
      this.size++;
    }
  }
  // Перевірка наявності
  has(value) {
    const index = this.hash(value);
    const bucket = this.buckets[index];
    return bucket && bucket.some(item => this.isEqual(item[0], value));
  }
  // Порівняння за SameValueZero
  isEqual(a, b) {
    if (a === b) return true;
    if (Number.isNaN(a) && Number.isNaN(b)) return true;
    if (a === 0 && b === 0) return true;
    return false;
  }
}
```

```
// Використання
const set = new SimpleSet();

// Додаємо значення, які викликають колізію
set.add("cat"); // Довжина "cat" = 3 → hash = 3 % 4 = 3
set.add("dog"); // Довжина "dog" = 3 → hash = 3 % 4 = 3 (колізія!)
set.add("bird"); // Довжина "bird" = 4 → hash = 4 % 4 = 0
set.add("fish"); // Довжина "fish" = 4 → hash = 4 % 4 = 0 (колізія!)
```

### Хеш-таблиця:

+-----+		
buckets (масив із 4 бакетів)		
-----		
Index 0	-> [	["bird", true]
		["fish", true] ]
-----		
Index 1	->	empty
-----		
Index 2	->	empty
-----		
Index 3	-> [	["cat", true]
		["dog", true] ]
+-----+		



## WeakMap

Вона дозволяє використовувати лише об'єкти як ключі, утримує слабкі посилання на ці ключі (дозволяючи збирачу сміття видаляти їх, якщо немає інших посилань)

WeakMap :

- це подібна до Map колекція, яка дозволяє використовувати *лише об'єкти, як ключі*
- значення з ключами-об'єктами автоматично видаляються, коли їх адреса більше ніде не зберігається
- має лише методи : `weakMap.get(key)`, `weakMap.set(key, value)`, `weakMap.delete(key)`, `weakMap.has(key)`
- відсутності властивості/методи: `clear`, `size`, `keys`, `values` ...
- не є ітерованим

```
//Створюємо об'єкт
let john = { name: "Іван" }

//Створюємо об'єкт WeakMap
let weakMap = new WeakMap();

//Додаємо елемент з ключем-об'єктом john
weakMap.set(john, "some data");

// перезапишемо посилання
john = null;

//В результаті john автоматично буде видалено з weakMap і видалено з пам'яті!
```

*Приклади використання:*

- Веб-додатки: Кешування результатів обробки даних користувачів.
- Серверні додатки: Зберігання метаданих для сесій або клієнтів, які можуть бути видалені.
- ...

## WeakSet

WeakSet :

- це подібна до Set колекція, яка зберігає тільки об'єкти
- значення-об'єкти автоматично видаляються, якщо їх адреса більше ніде не зберігається
- має методи: `add`, `has` і `delete`
- відсутності властивості/методи: `clear`, `size`, `keys`, `values` ...
- не є ітерованим

```
//Створюємо об'єкти
```

```
let john = { name: "Іван" };
```

```
let peter = { name: "Петро" };
```

```
//створюємо об'єкт WeakSet
```

```
let visitedSet = new WeakSet();
```

```
//додаємо об'єкти до слабкої множини
```

```
visitedSet.add(john);
```

```
visitedSet.add(peter);
```

```
john = null;
```

```
// john буде автоматично видалено з visitedSet та видалено з пам'яті
```



