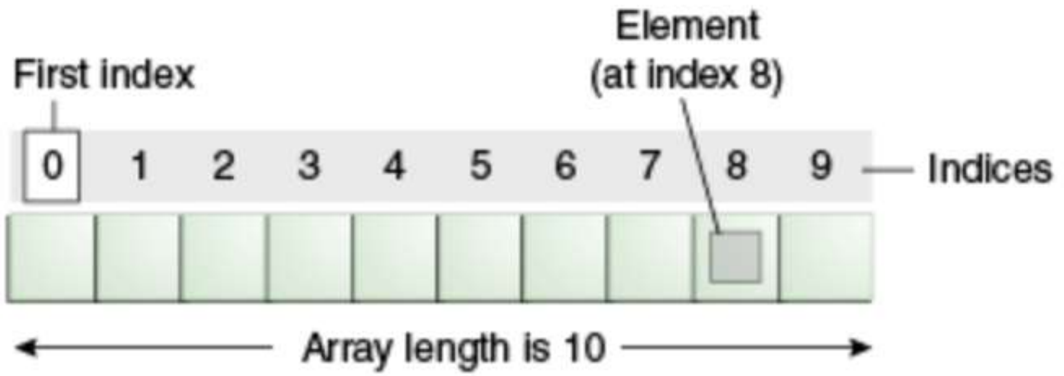
sourcemind

ARRAYS IN DEPTH



An array of 10 elements.

- **JavaScript arrays are resizable** and **can contain a mix of different data types**. (When those characteristics are undesirable, use typed arrays instead.)
- **JavaScript arrays are not associative arrays** and so, array elements cannot be accessed using arbitrary strings as indexes, but must be accessed using nonnegative integers (or their respective string form) as indexes.
- **JavaScript arrays are zero-indexed**: the first element of an array is at index `0`, the second is at index `1`, and so on — and the last element is at the value of the array's `length` property minus `1`.
- **JavaScript array-copy operations create shallow copies**. (All standard built-in copy operations with *any* JavaScript objects create shallow copies, rather than deep copies).

sourcemind

```
let arr = new Array();
let arr = [];
```

```
let fruits = ["Apple", "Orange", "Plum"];
```

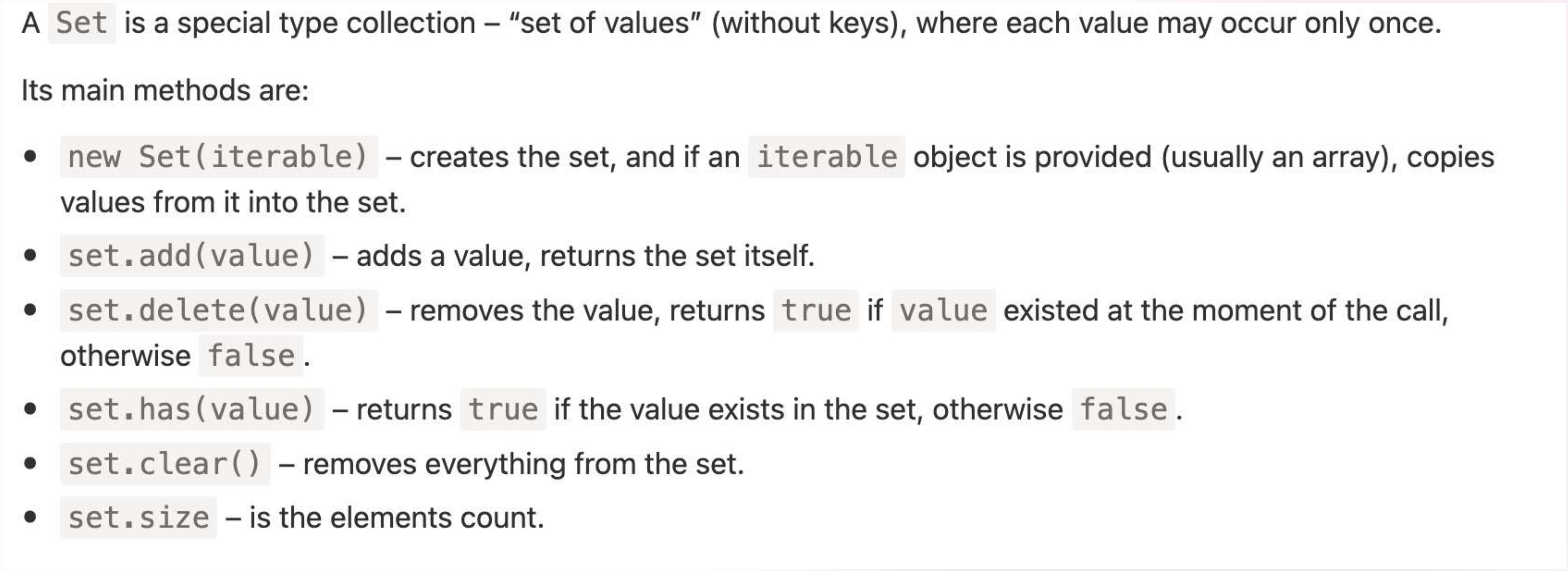
Nested Arrays

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

JavaScript Array Methods

- pop()
- push()
- toString()
- join()
- splice()
- sort()
- shift()
- unshift()
- reverse()
- concat()
- slice()
- filter()
- find()
- forEach()
- map()
- reduce()
- every()
- some()

Set



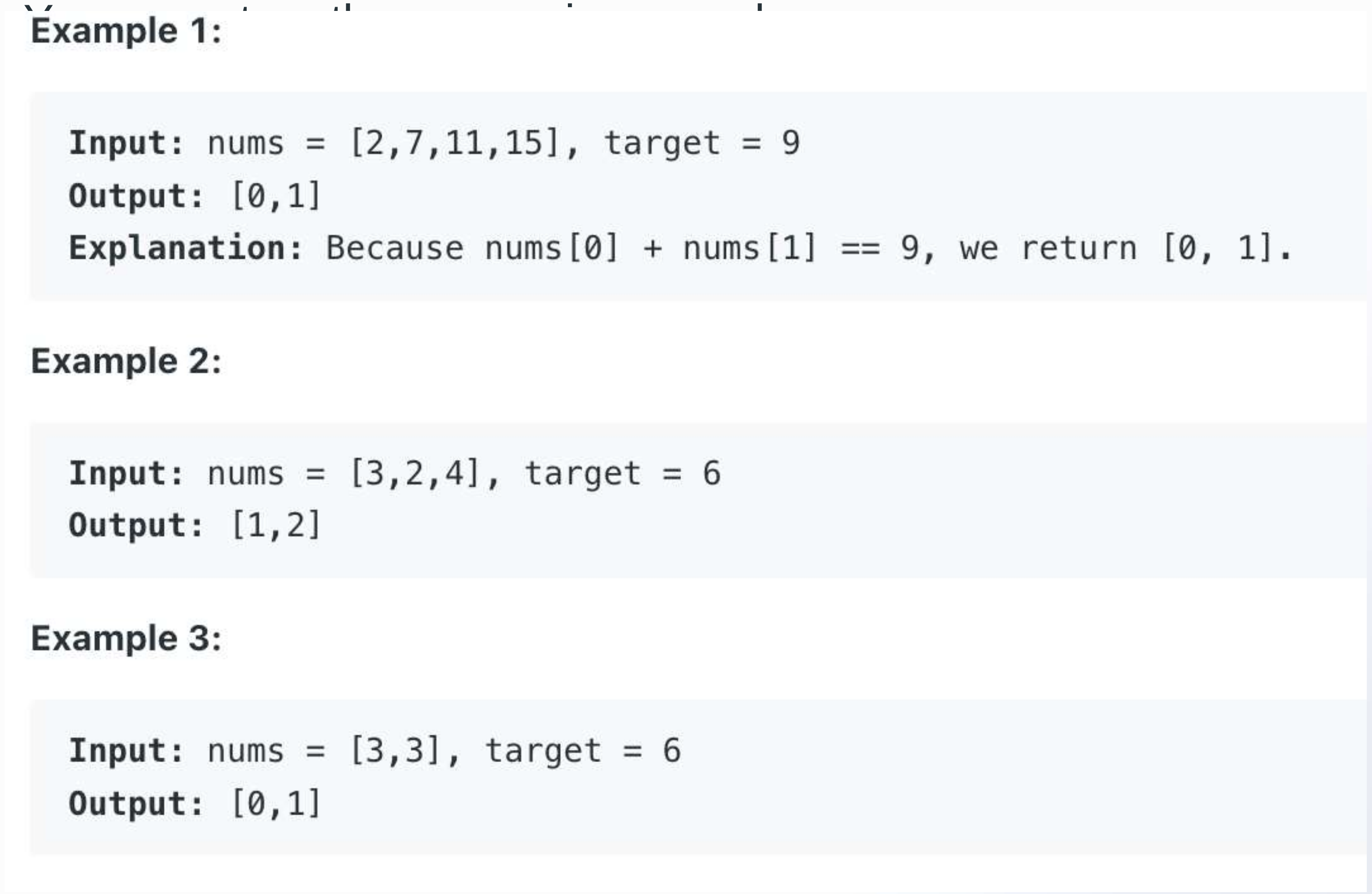
A `Set` is a special type collection – “set of values” (without keys), where each value may occur only once.

Its main methods are:

- `new Set(iterable)` – creates the set, and if an `iterable` object is provided (usually an array), copies values from it into the set.
- `set.add(value)` – adds a value, returns the set itself.
- `set.delete(value)` – removes the value, returns `true` if `value` existed at the moment of the call, otherwise `false`.
- `set.has(value)` – returns `true` if the value exists in the set, otherwise `false`.
- `set.clear()` – removes everything from the set.
- `set.size` – is the elements count.

Task

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.
You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.
[illegible]



Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

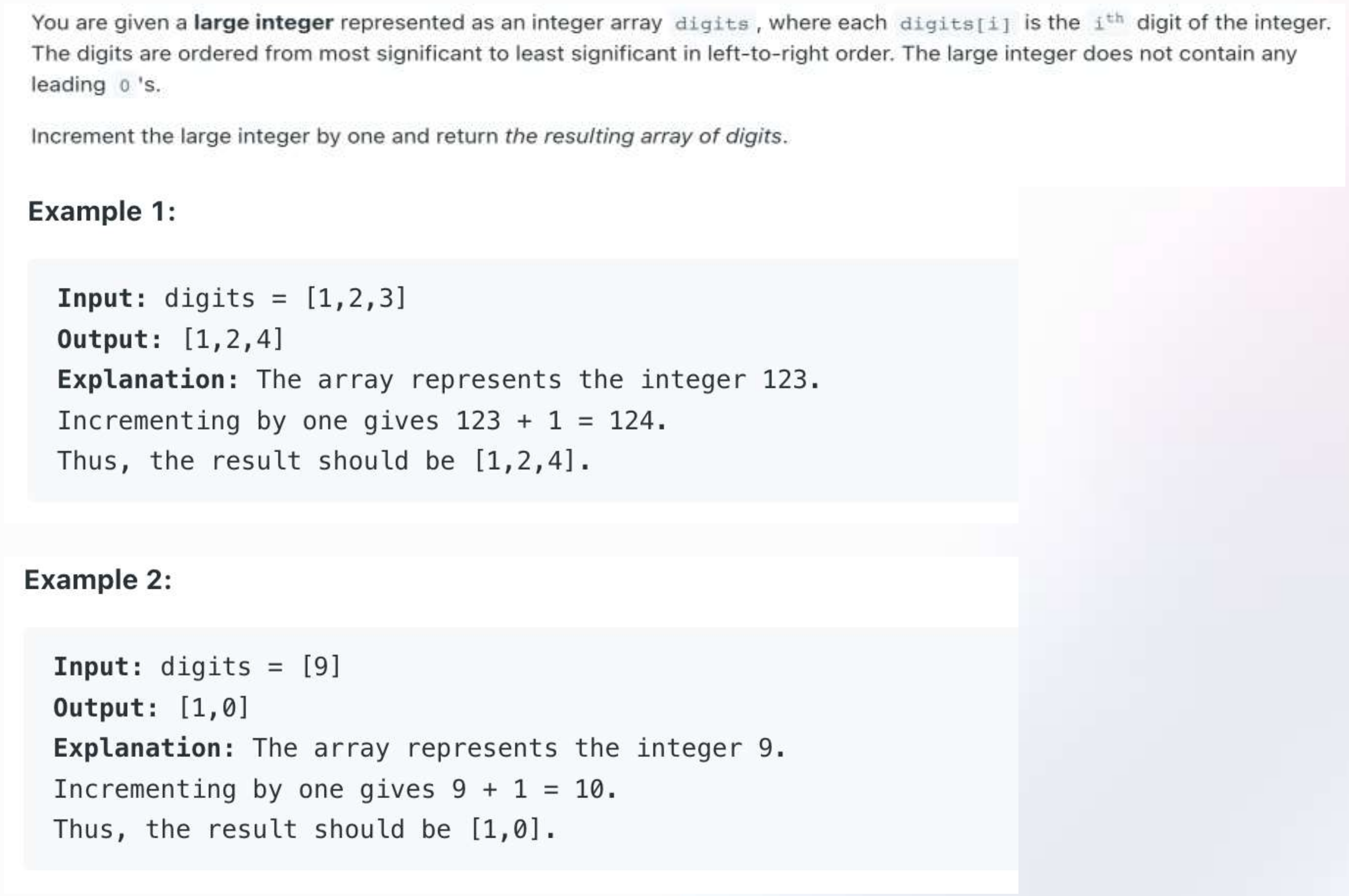
Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Task



You are given a **large integer** represented as an integer array `digits`, where each `digits[i]` is the `i`th digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading `0`'s.

Increment the large integer by one and return *the resulting array of digits*.

Example 1:

```
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].
```

Example 2:

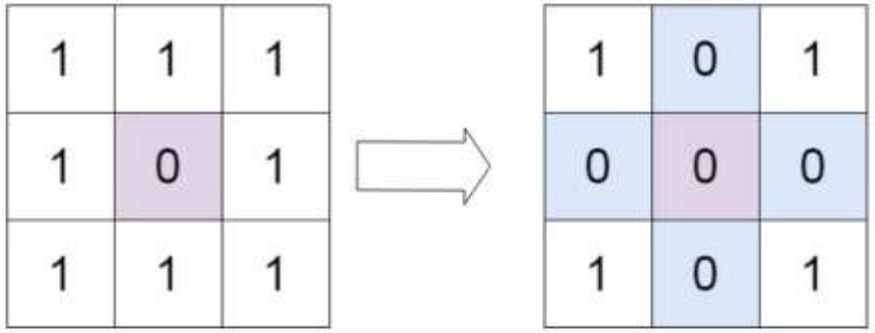
```
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].
```

Task

Given an `m x n` integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

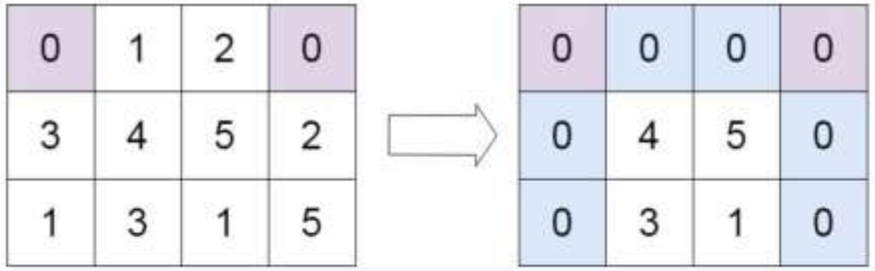
You must do it in place.

Example 1:



```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

Example 2:



```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```