A COMPREHENSIVE GUIDE TO JAVASCRIPT ARRAY METHODS

Categorized by purpose for easy understanding.

Adding and Removing Elements
Methods to add or take away items from the start, end, or middle.

Iterating and Transforming
Functional methods to loop through arrays and manipulate data.

Searching and Finding
Methods to locate specific items or indices within an array.

Slicing and Joining
Methods to combine arrays or cut them into smaller pieces.

Ordering
Methods to change the arrangement of elements.

Testing Conditions
Methods that return boolean true/false based on array contents.

Utility & Static Methods
General-purpose and static methods called on the Array constructor.

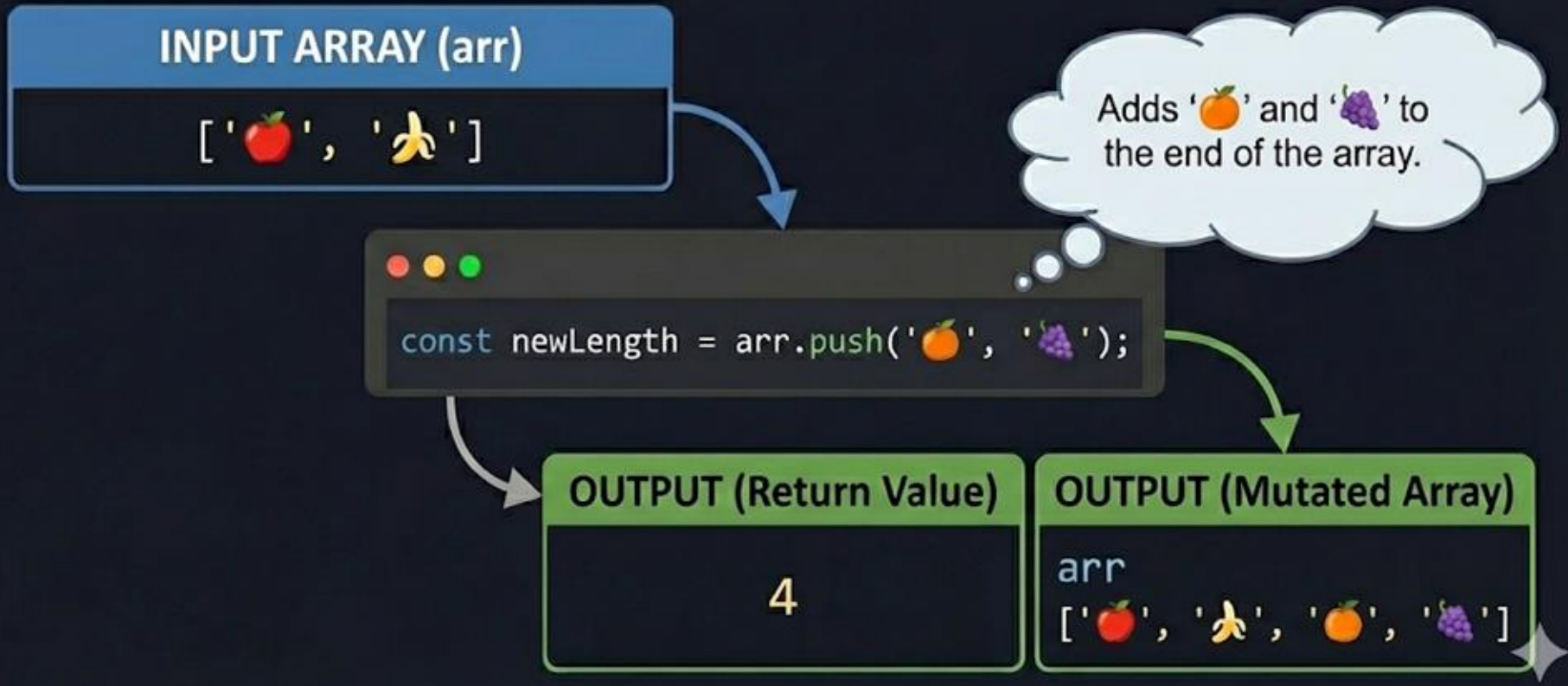
Includes Mutating (modifies original) and Non-Mutating (returns new value) methods.

ADDING AND REMOVING ELEMENTS

- push() (Mutating) 
- pop() (Mutating) 
- unshift() (Mutating) 
- shift() (Mutating) 
- splice() (Mutating) 

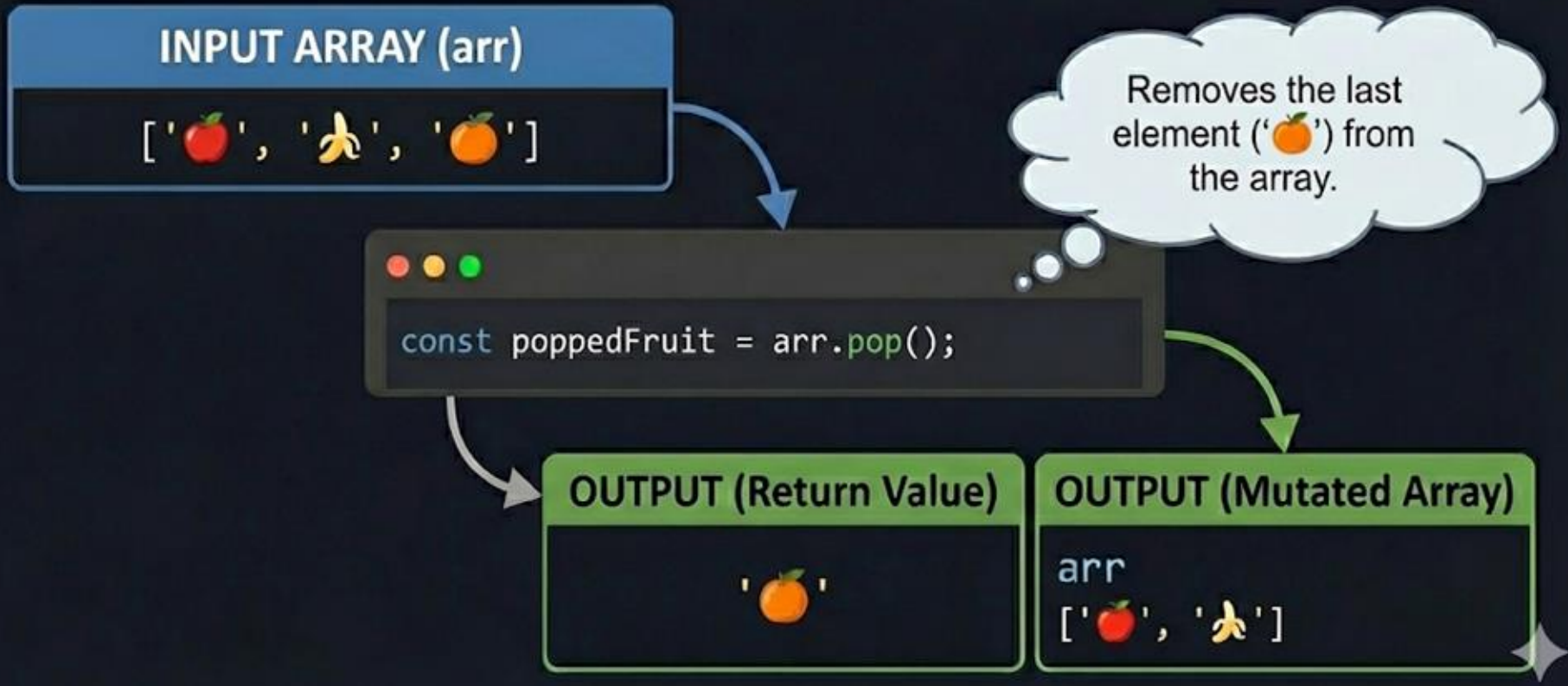
Array.prototype.push() (Mutating)

Adds one or more elements to the end of an array and returns the new length of the array.



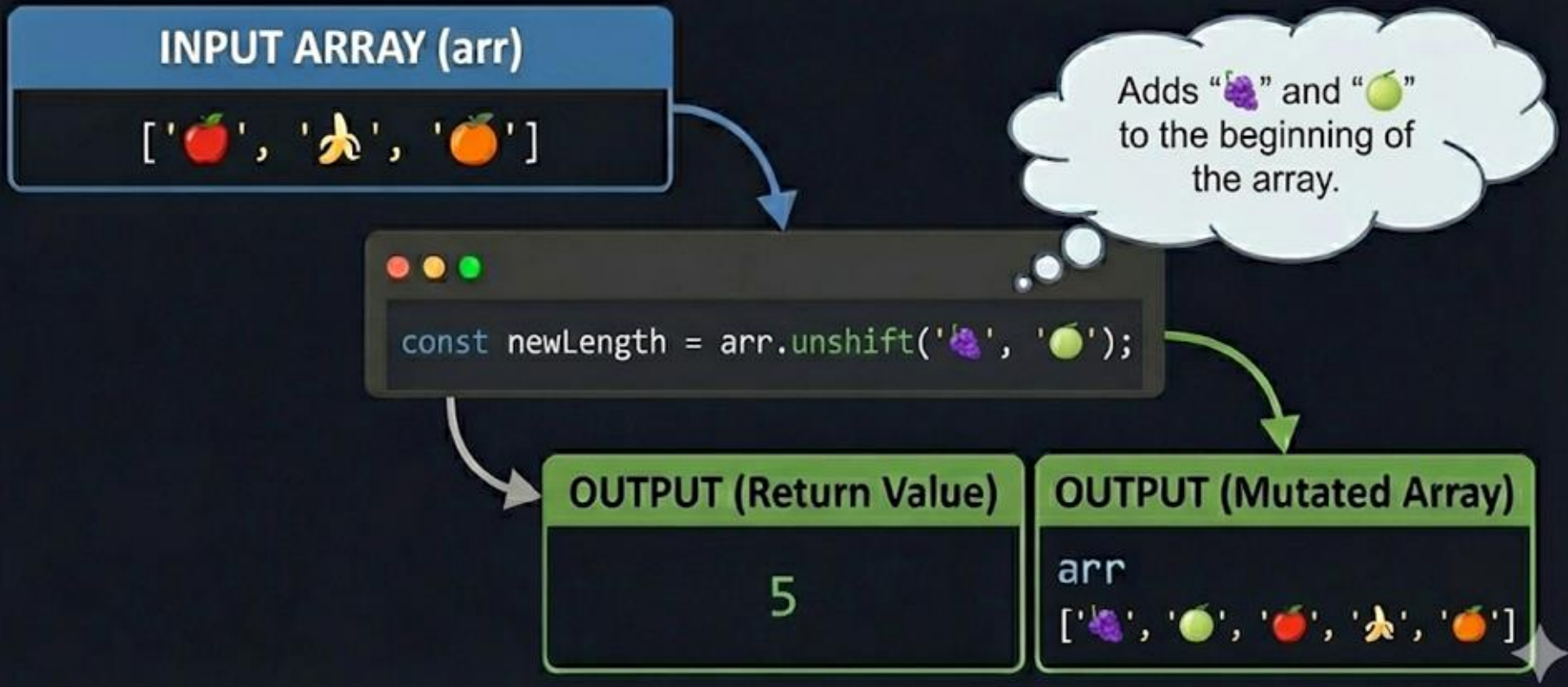
Array.prototype.pop() (Mutating)

Removes the last element from an array and returns that element.



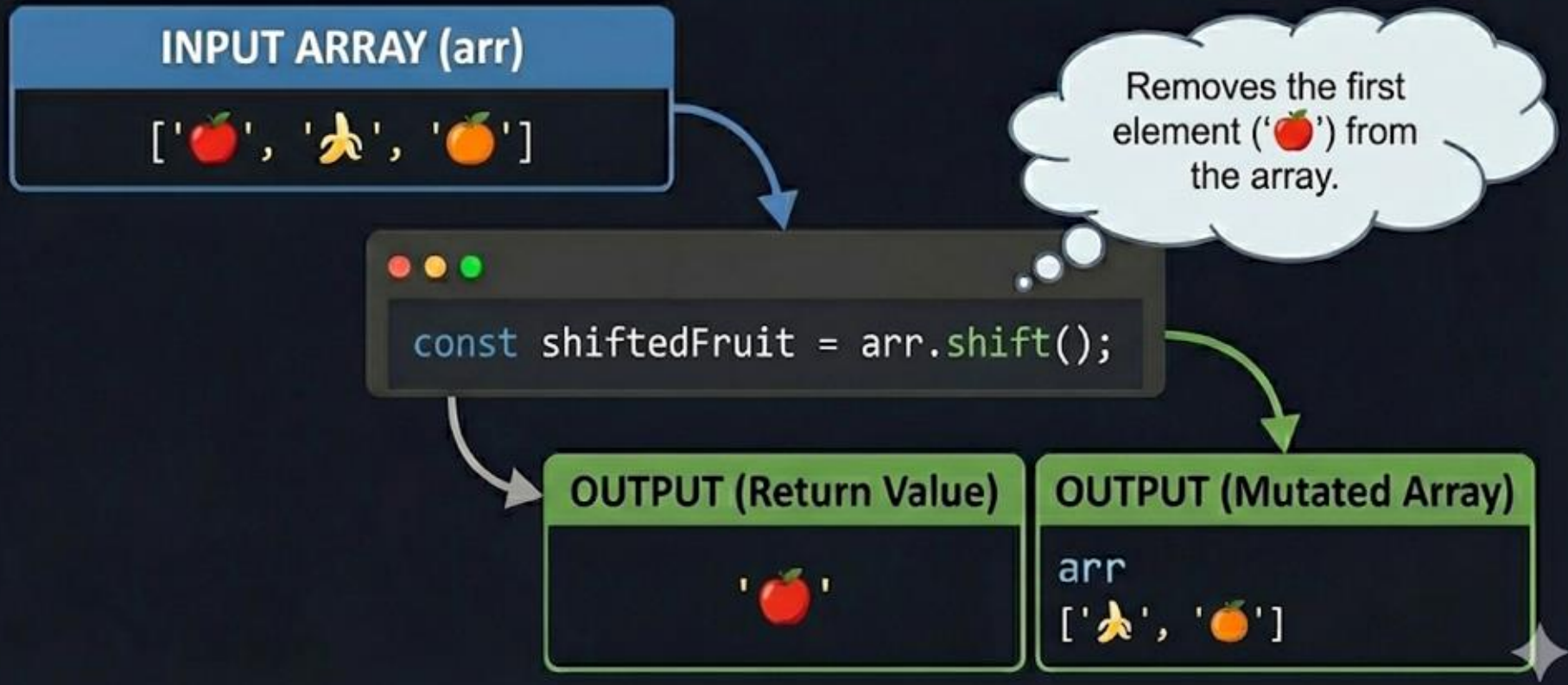
Array.prototype.unshift() (Mutating)

Adds one or more elements to the beginning of an array and returns the new length of the array.



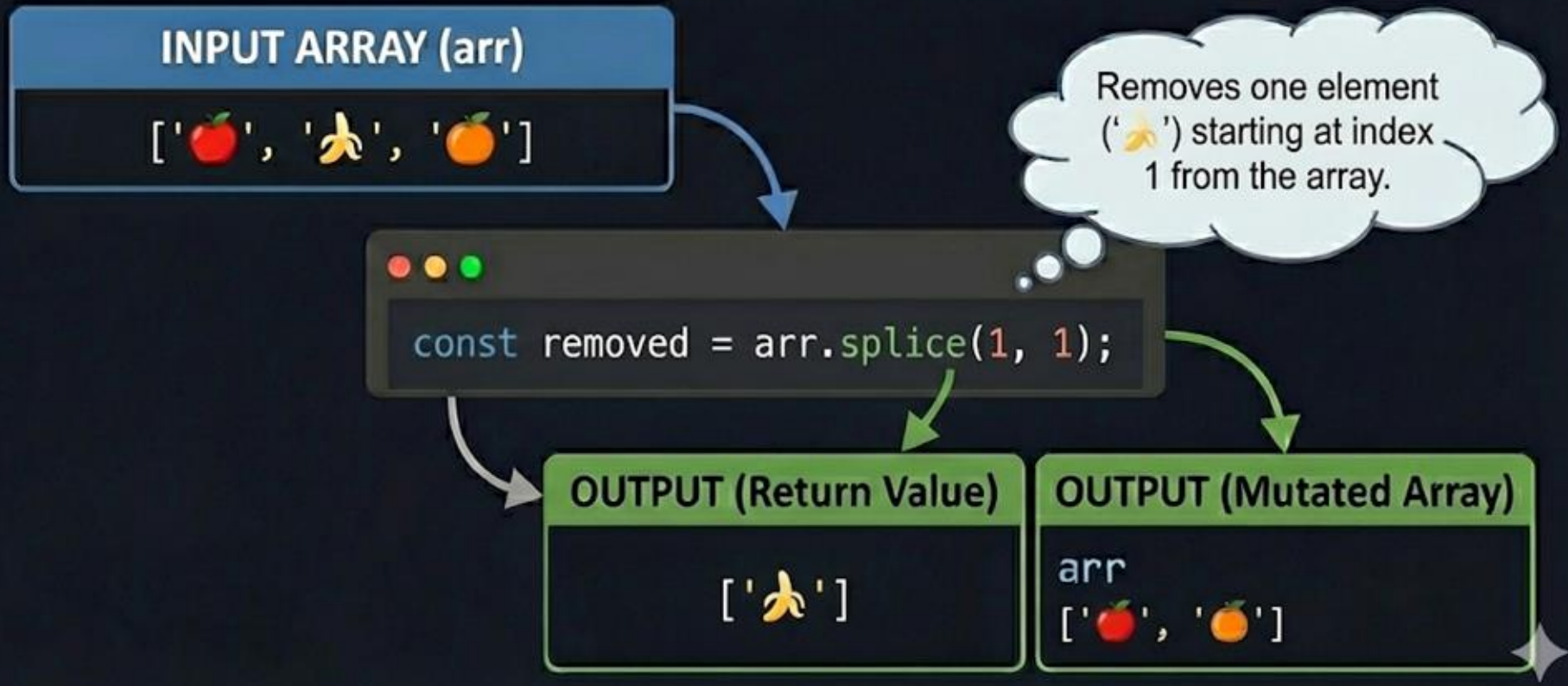
Array.prototype.shift() (Mutating)

Removes the first element from an array and returns that element.



Array.prototype.splice() (Mutating)

Adds or removes elements from an array. In this example, we remove one element starting at index 1.

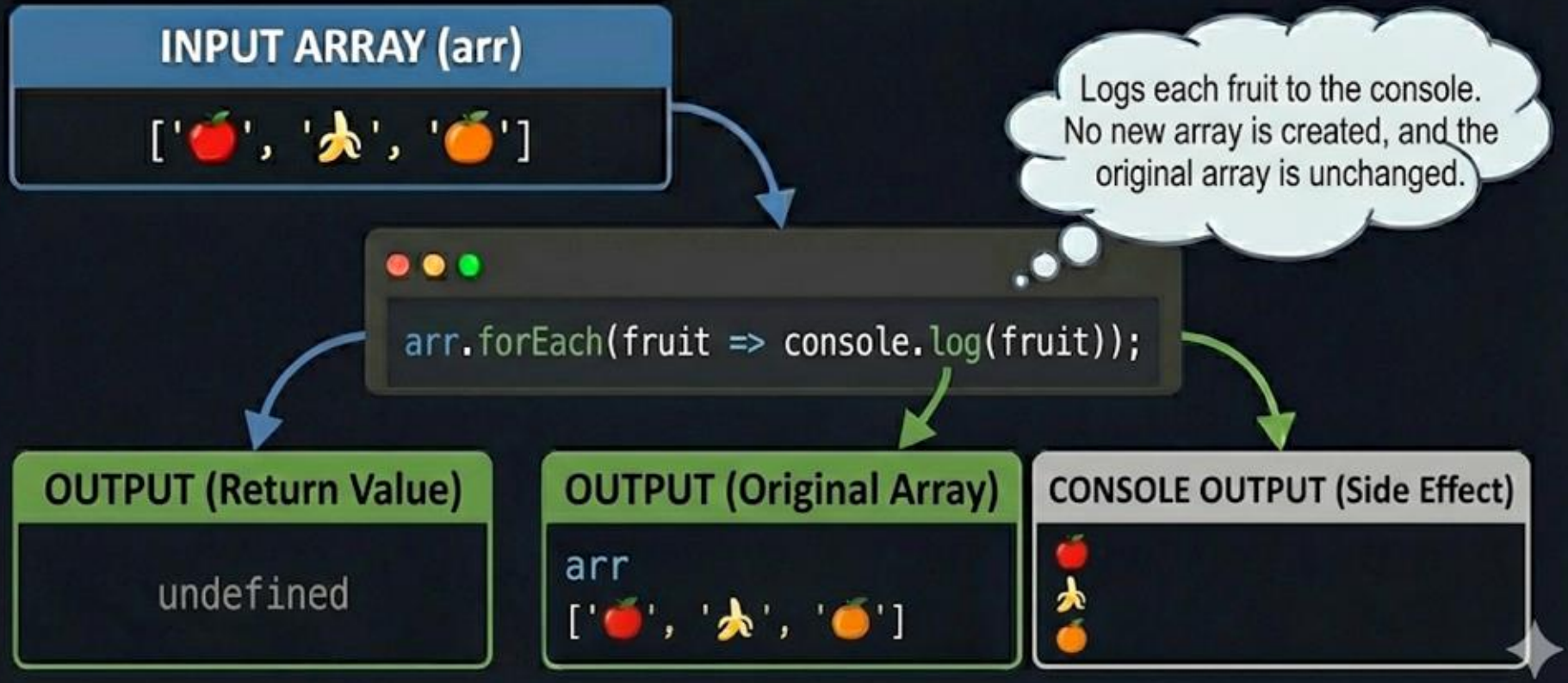


ITERATING AND TRANSFORMING

- forEach() (Non-Mutating*) 
- map() (Non-Mutating) 
- filter() (Non-Mutating) 
- reduce() (Non-Mutating) 
- reduceRight() (Non-Mutating) 
- flat() (Non-Mutating) 
- flatMap() (Non-Mutating) 

Array.prototype.forEach() (Non-Mutating*)

Executes a provided function once for each array element. This is commonly used for side effects, like logging to the console, as shown here.



Array.prototype.map() (Non-Mutating)

Creates a new array populated with the results of calling a provided function on every element in the calling array. In this example, we transform each fruit into a different colored circle representing its color.

INPUT ARRAY (arr)

['🍎', '🍌', '🍊']

Transforms each fruit into a colored circle based on its color.

```
const newArr = arr.map(fruit => {
  if (fruit === '🍎') return '🔴';
  else if (fruit === '🍌') return '🟡';
  else{
    return '🟠';
  }
});
```

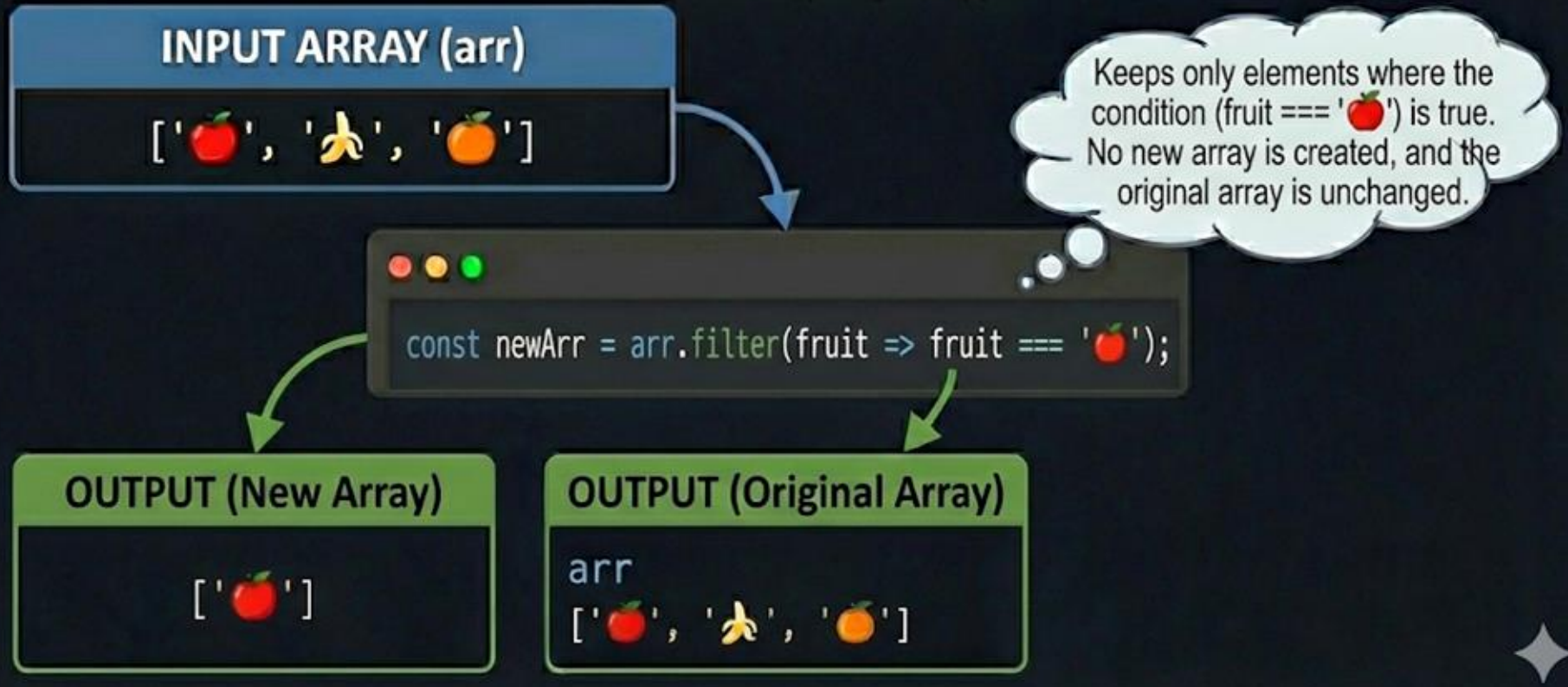
OUTPUT (New Array)

newArr

['🔴', '🟡', '🟠']

Array.prototype.filter() (Non-Mutating)

Creates a new array with all elements that pass the test implemented by the provided function.
Here, we filter to keep only the apple.



Array.prototype.reduce() (Non-Mutating)

Executes a 'reducer' function on each element of the array, resulting in a single output value.
In this example, we sum the lengths of the fruit names ('apple' is 5, 'banana' is 6, 'orange' is 6, total is 17).

INPUT ARRAY (arr)

['🍎', '🍌', '🍊']

Iterates and accumulates the length of each fruit name to a single value ('apple' + 'banana' + 'orange' = 5+6+6 = 17).

```
const totalLength = arr.reduce((acc, fruit) => acc + fruit.length, 0);
```

OUTPUT (Single Value)

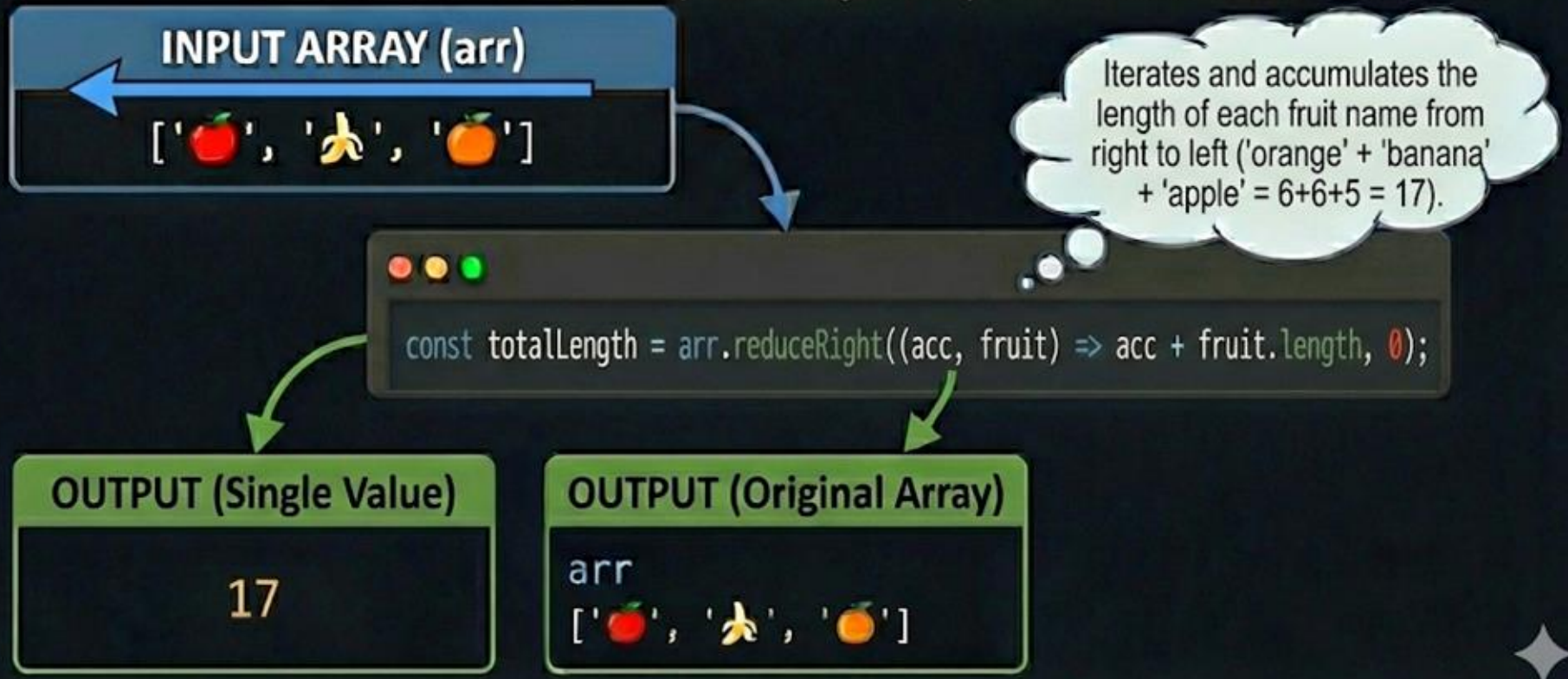
17

OUTPUT (Original Array)

arr
['🍎', '🍌', '🍊']

Array.prototype.reduceRight() (Non-Mutating)

This method works exactly like **reduce()**, but it processes the array elements from right-to-left. The arrow above the input array in the image visually indicates this direction.



Array.prototype.flat() (Non-Mutating)

Creates a new array with all sub-array elements concatenated into it recursively.
Here, a nested array ['🍎', ['🍌', '🍊']] is flattened into a single-level array.

INPUT ARRAY (arr)

['🍎', ['🍌', '🍊']]

Flatten the nested array. The inner array ['🍌', '🍊'] is concatenated into the main array.

```
const flattenedArr = arr.flat();
```

OUTPUT (New Array)

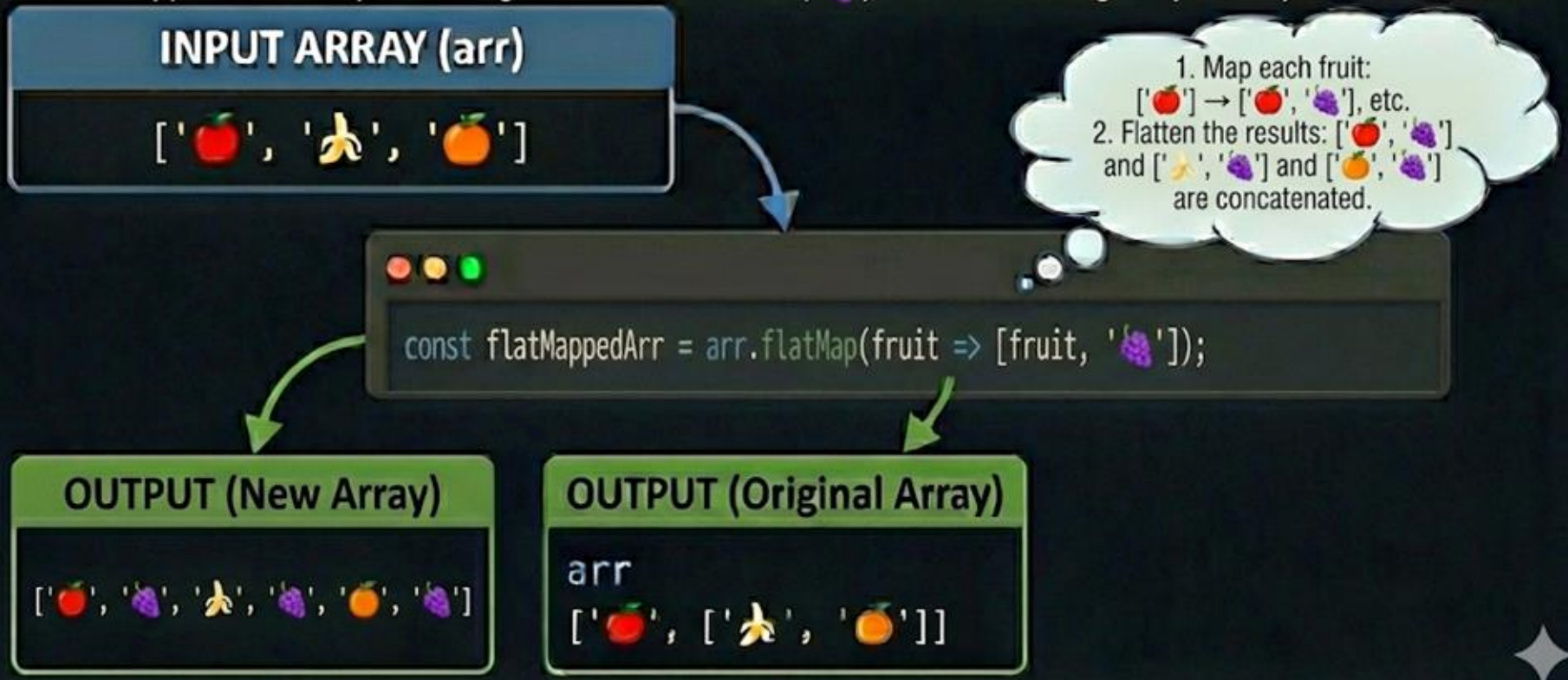
['🍎', '🍌', '🍊']

OUTPUT (Original Array)

arr
['🍎', ['🍌', '🍊']]

Array.prototype.flatMap() (Non-Mutating)

First maps each element using a mapping function, then flattens the result into a new array. In this example, each fruit is mapped to an array containing itself and a new fruit ('🍇'), and the resulting array of arrays is then flattened.



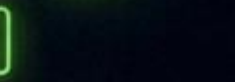
SEARCHING AND FINDING

- find() (Non-Mutating) 

- findIndex() (Non-Mutating) 

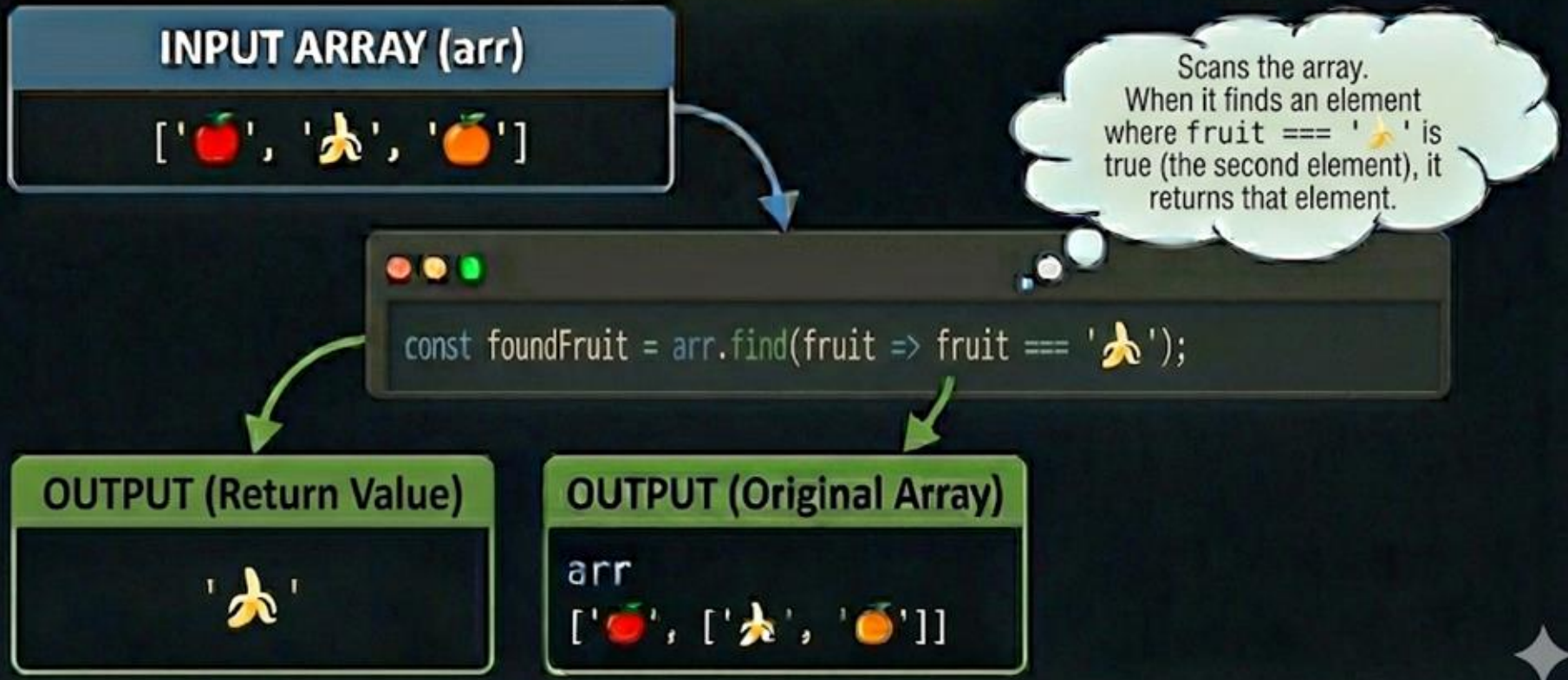
- indexOf() (Non-Mutating) 

- lastIndexOf() (Non-Mutating) 

- includes() (Non-Mutating) 
- at() (Non-Mutating) 

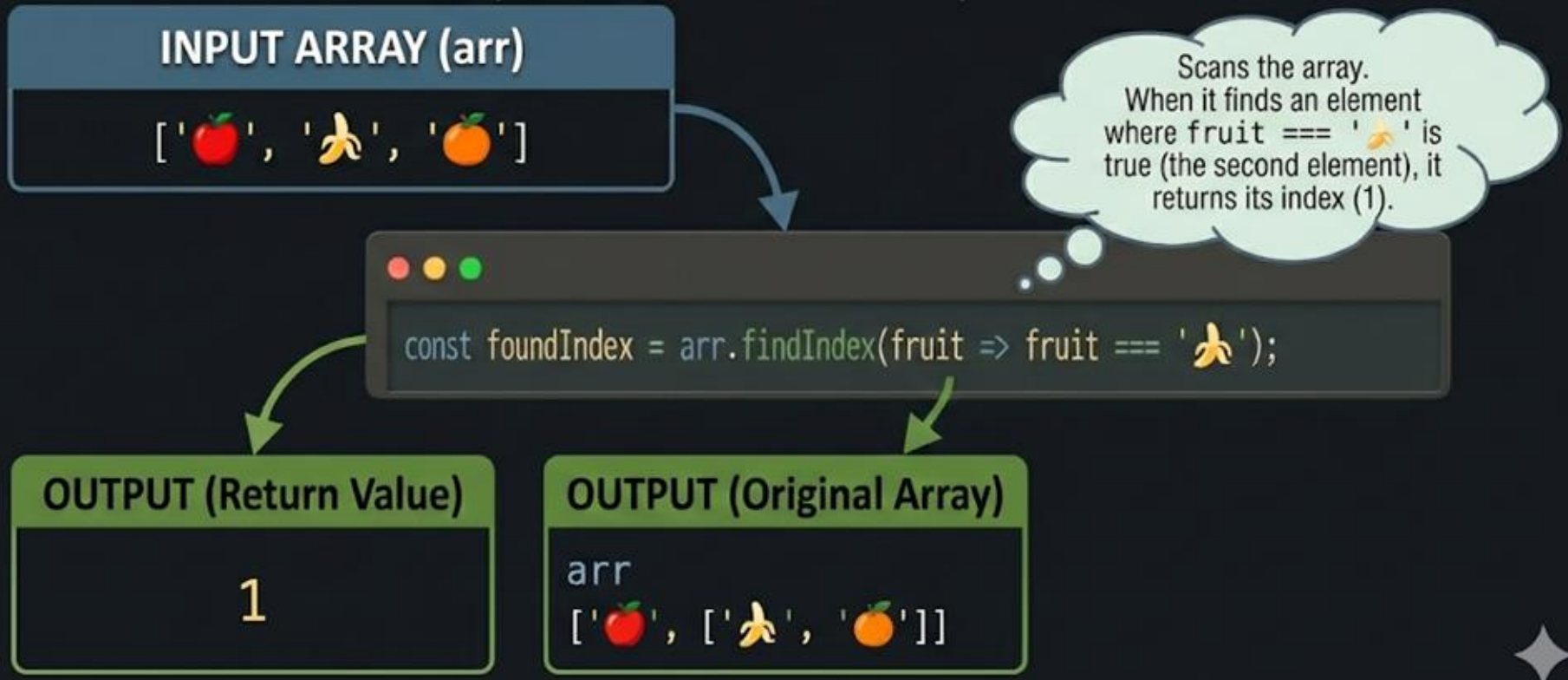
Array.prototype.find() (Non-Mutating)

Returns the value of the first element in the array that satisfies the provided testing function.
Here, we search for the banana.



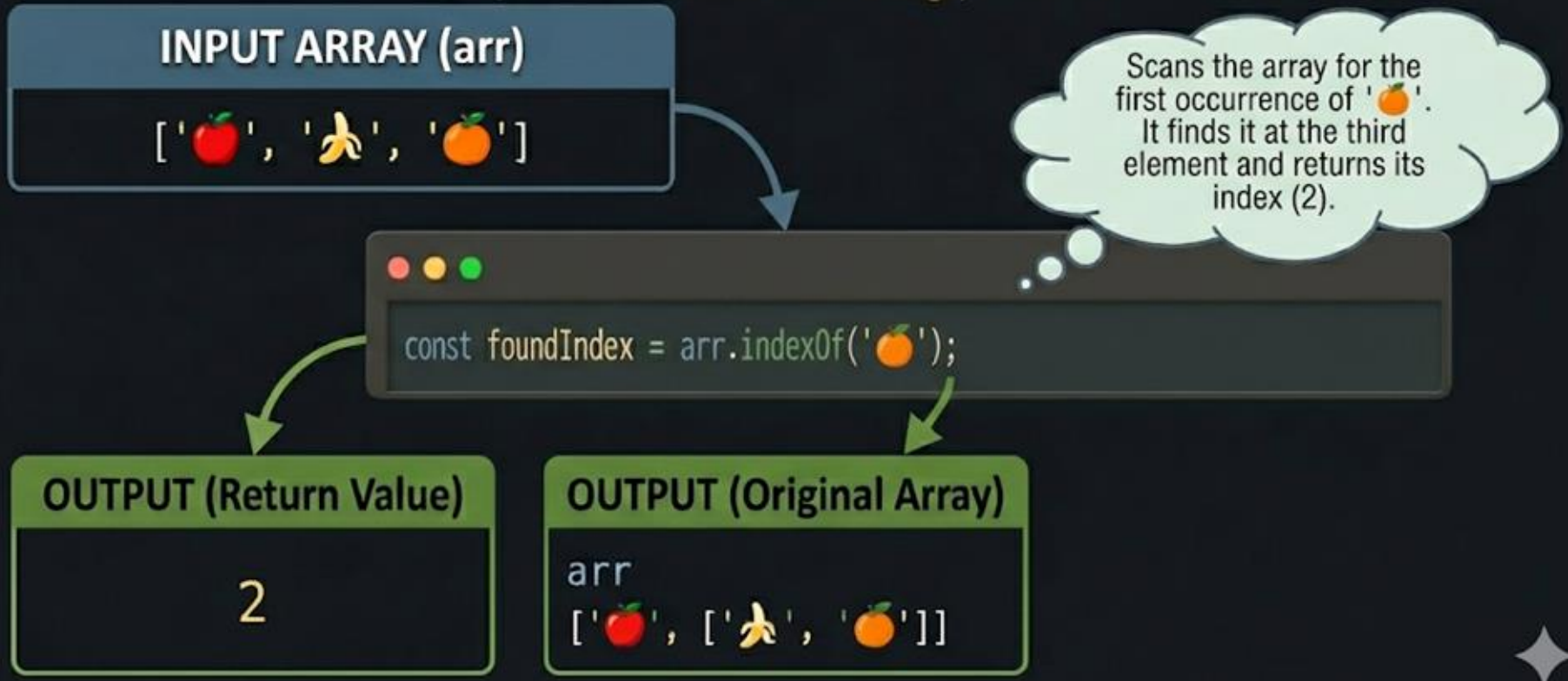
Array.prototype.findIndex() (Non-Mutating)

Returns the index of the first element in the array that satisfies the provided testing function.
Here, we find the index of the banana, which is 1.



Array.prototype.indexOf() (Non-Mutating)

Returns the first index at which a given element can be found in the array.
Here, we find the index of the orange, which is 2.



Array.prototype.lastIndexOf() (Non-Mutating)

Returns the last index at which a given element can be found in the array.
To demonstrate, we've added a second apple to the array. The last index of the apple is 2.

INPUT ARRAY (arr)

['🍎', '🍌', '🍎']

Scans the array from the end for the last occurrence of '🍎'. It finds it at the third element (index 2) and returns its index.

```
const lastIndex = arr.lastIndexOf('🍎');
```

OUTPUT (Return Value)

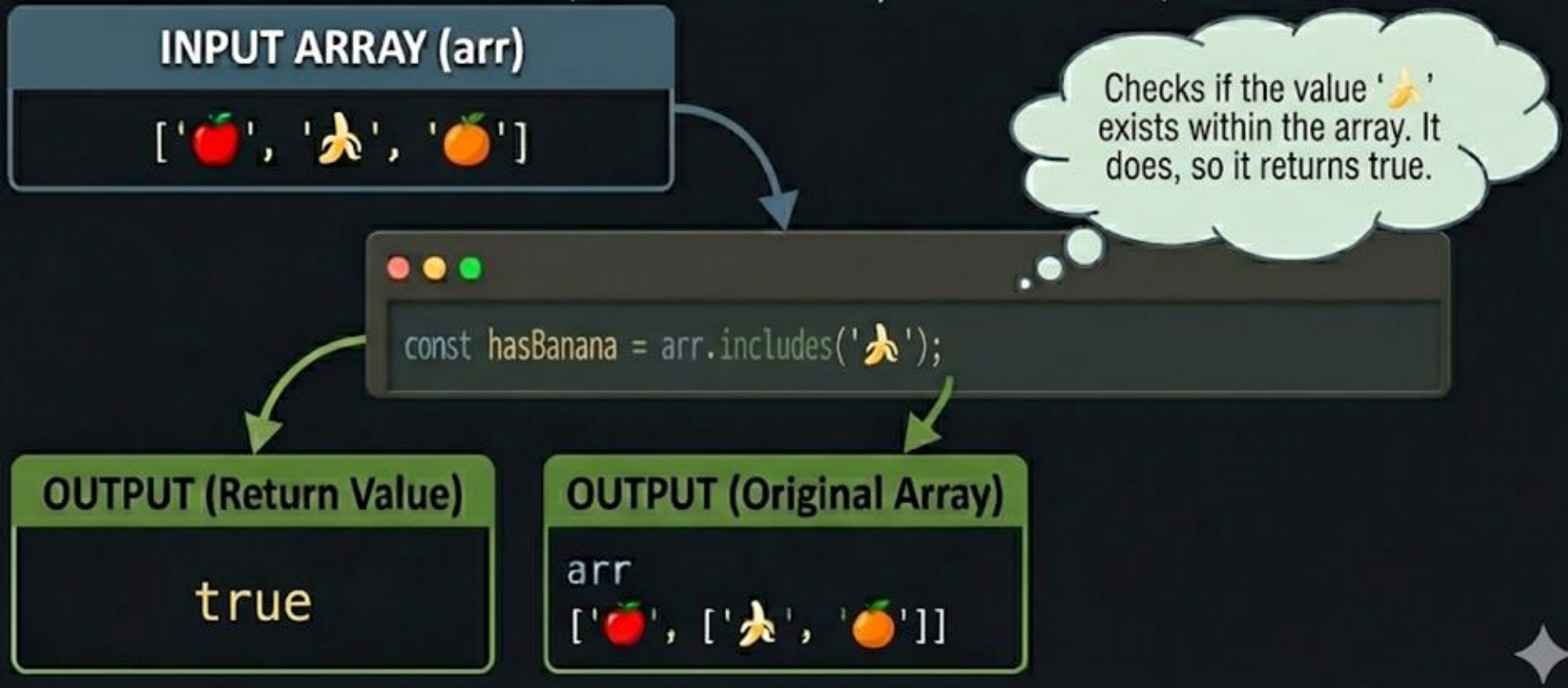
2

OUTPUT (Original Array)

arr
['🍎', ['🍌', '🍎']]

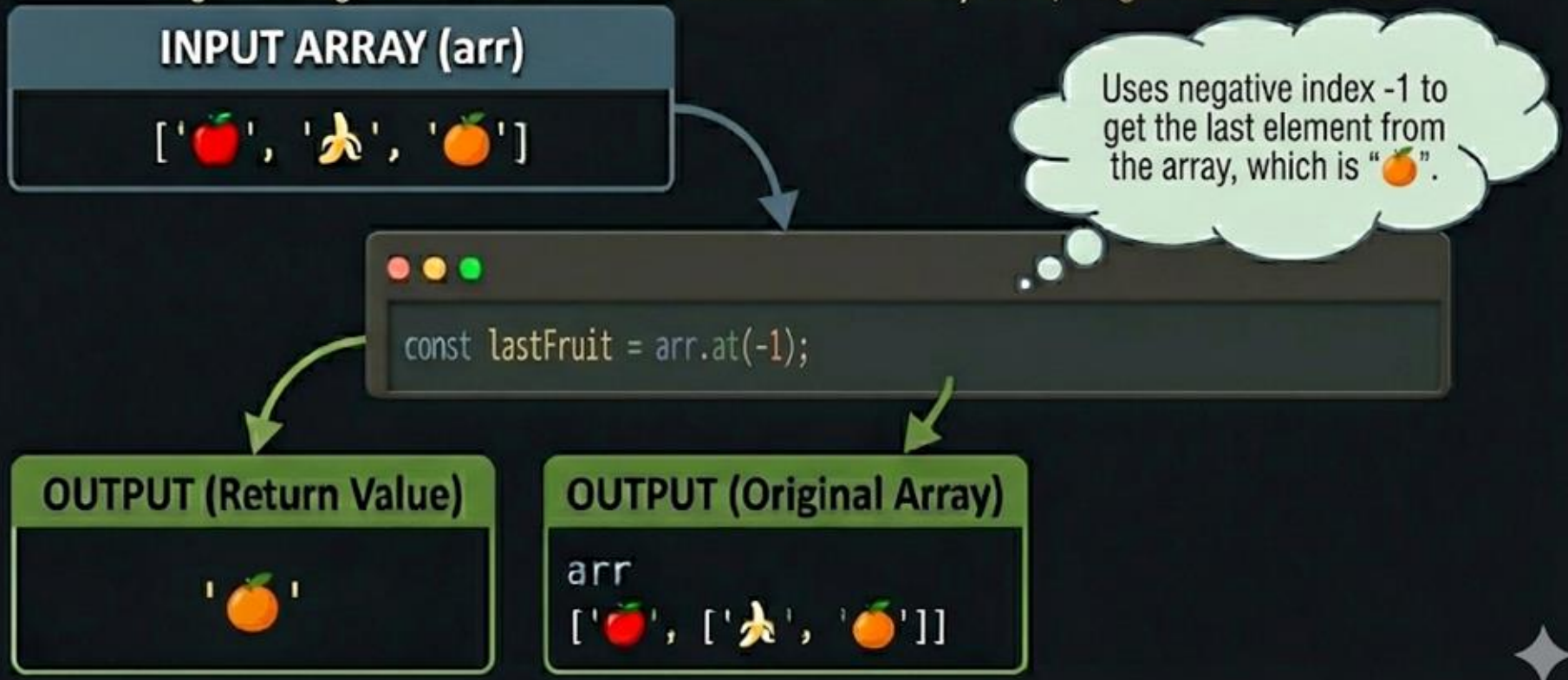
Array.prototype.includes() (Non-Mutating)

Determines whether an array includes a certain value among its entries, returning a boolean true or false. Here, we check if the array includes a banana, which is true.



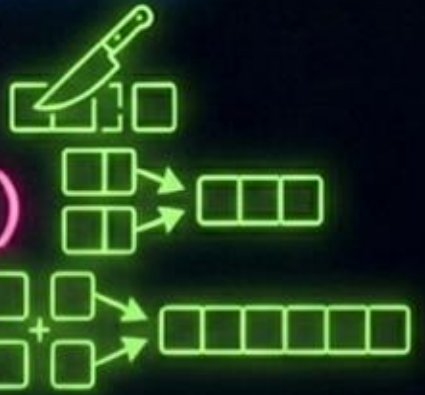
Array.prototype.at() (Non-Mutating)

Takes an integer value and returns the item at that index, allowing for positive and negative integers. Negative integers count back from the last item in the array. Here, we get the last element.



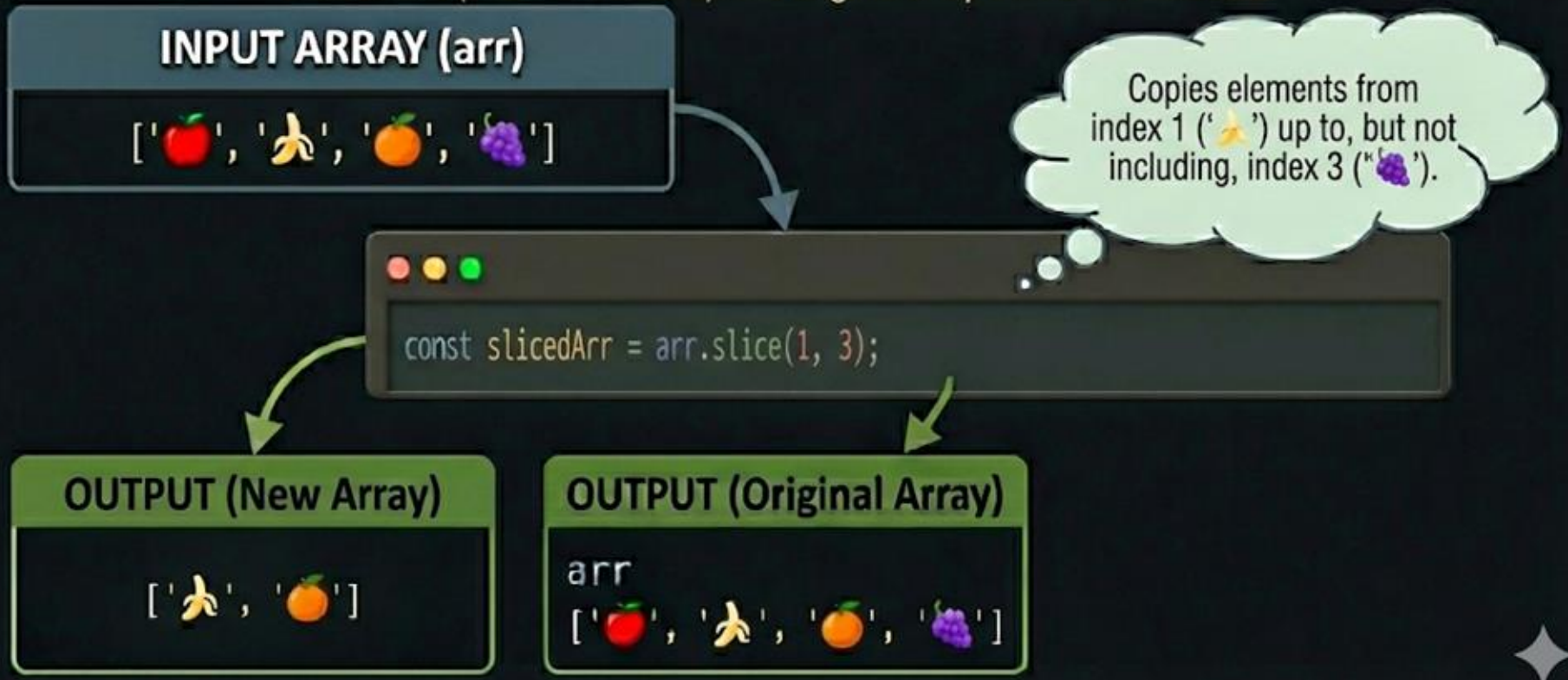
SLICING AND JOINING

- slice() (Non-Mutating)
- concat() (Non-Mutating)
- join() (Non-Mutating)



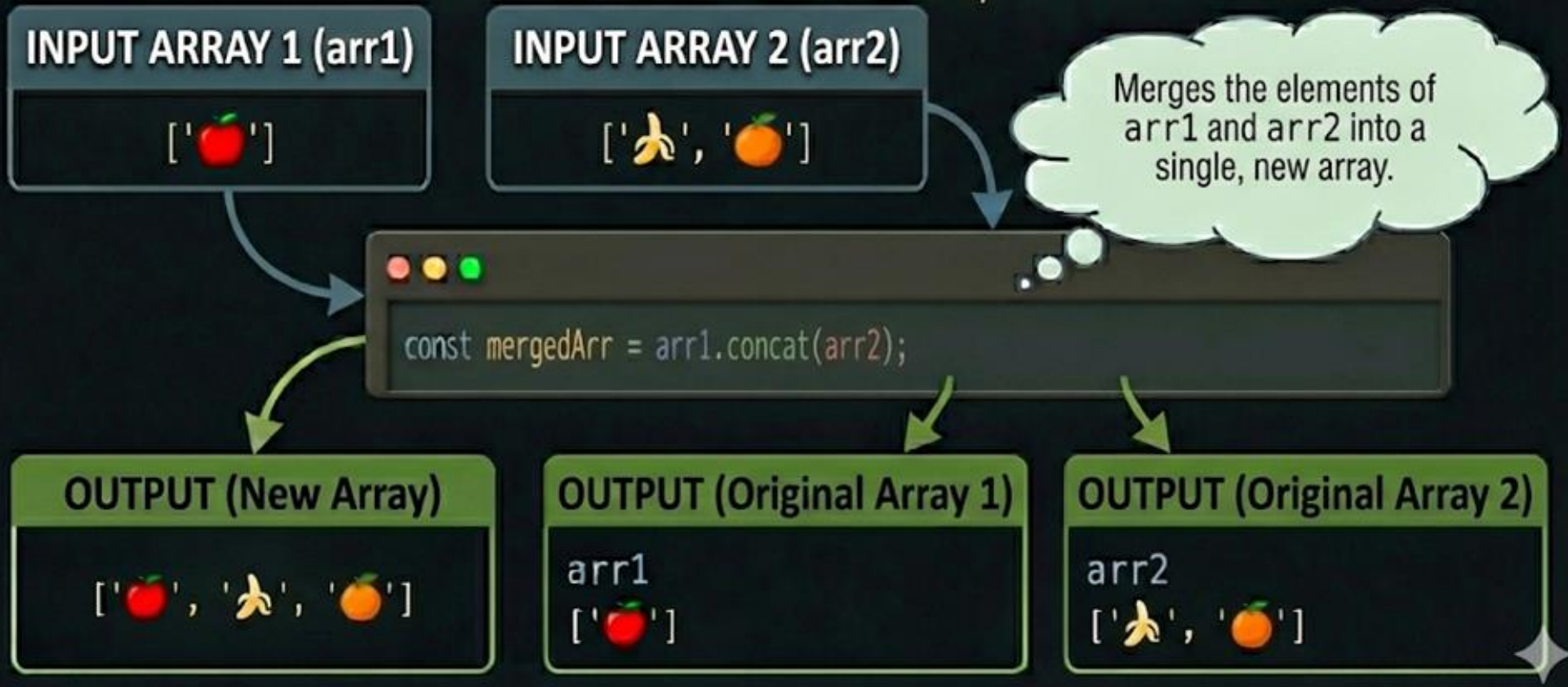
Array.prototype.slice() (Non-Mutating)

Returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included). The original array will not be modified.



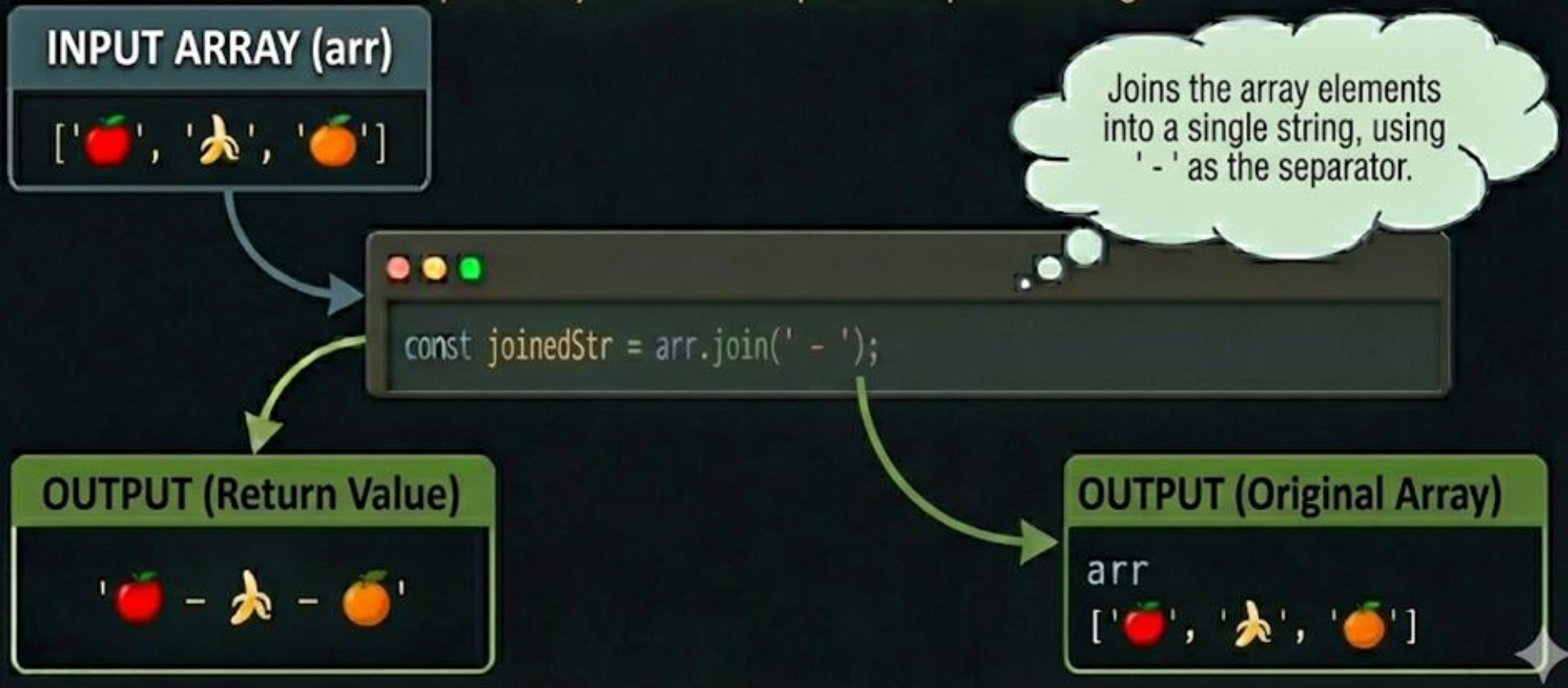
Array.prototype.concat() (Non-Mutating)

Used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.



Array.prototype.join() (Non-Mutating)

Creates and returns a new string by concatenating all of the elements in an array, separated by commas or a specified separator string.

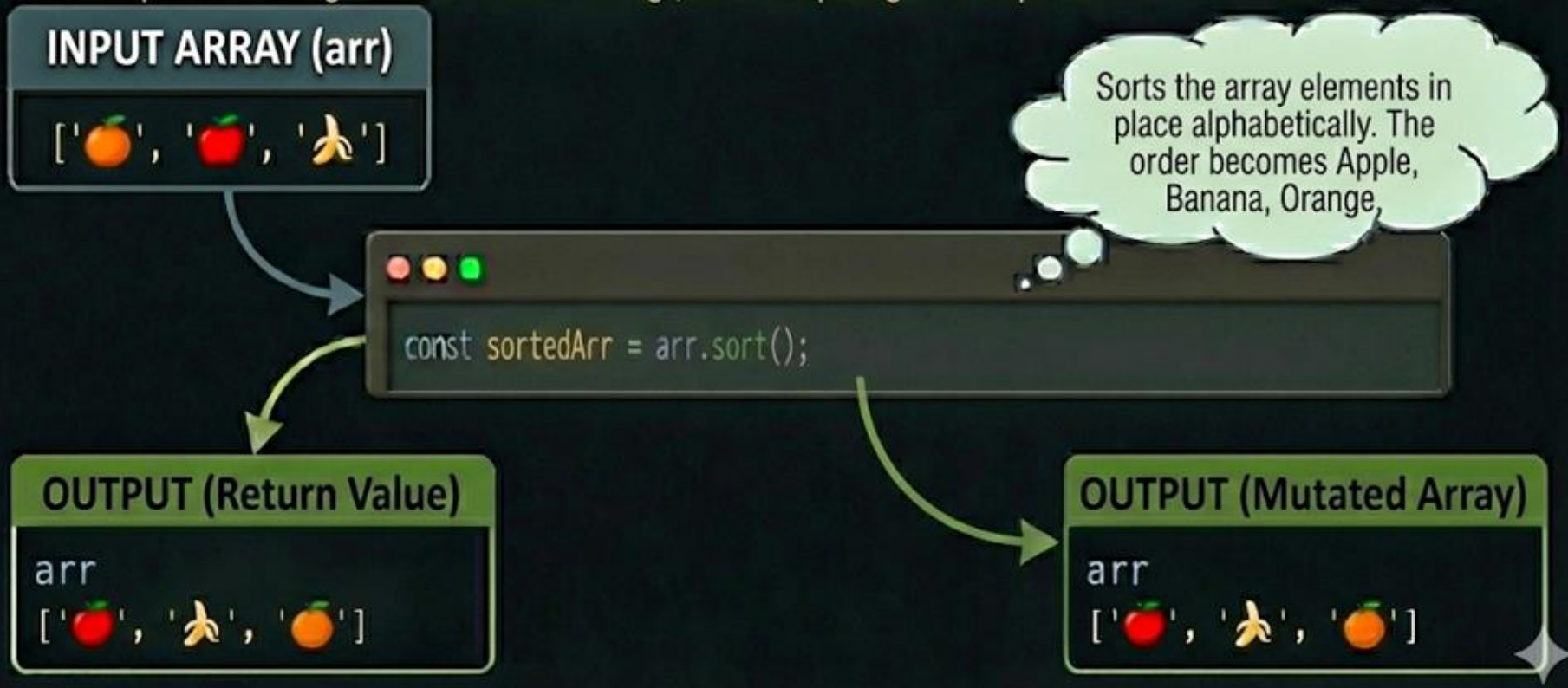


ORDERING

- sort() (Mutating) 
- reverse() (Mutating) 

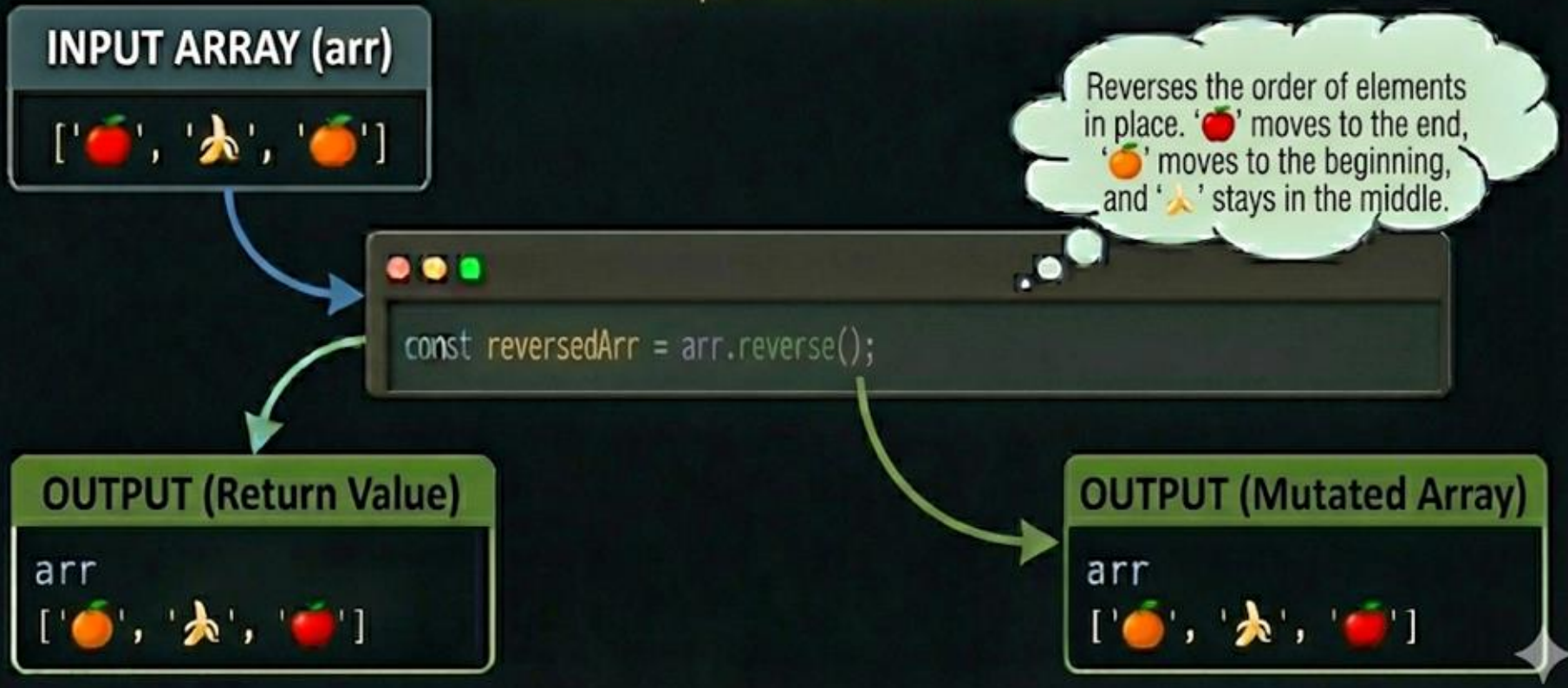
Array.prototype.sort() (Mutating)

Sorts the elements of an array in place and returns the sorted array. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.



Array.prototype.reverse() (Mutating)

Reverses an array in place. The first array element becomes the last, and the last array element becomes the first.

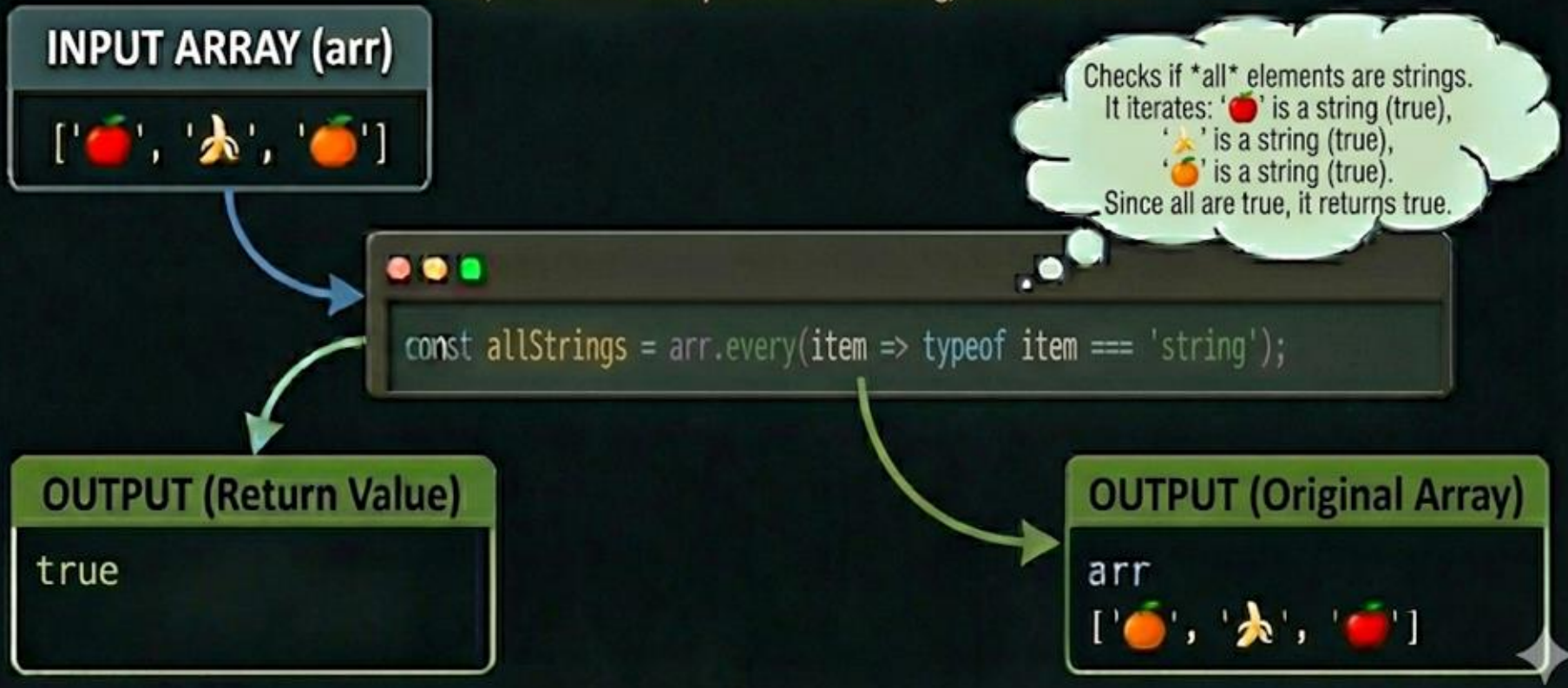


TESTING CONDITIONS

- every() (Non-Mutating) ☑☑☑
- some() (Non-Mutating) ☐☑☐

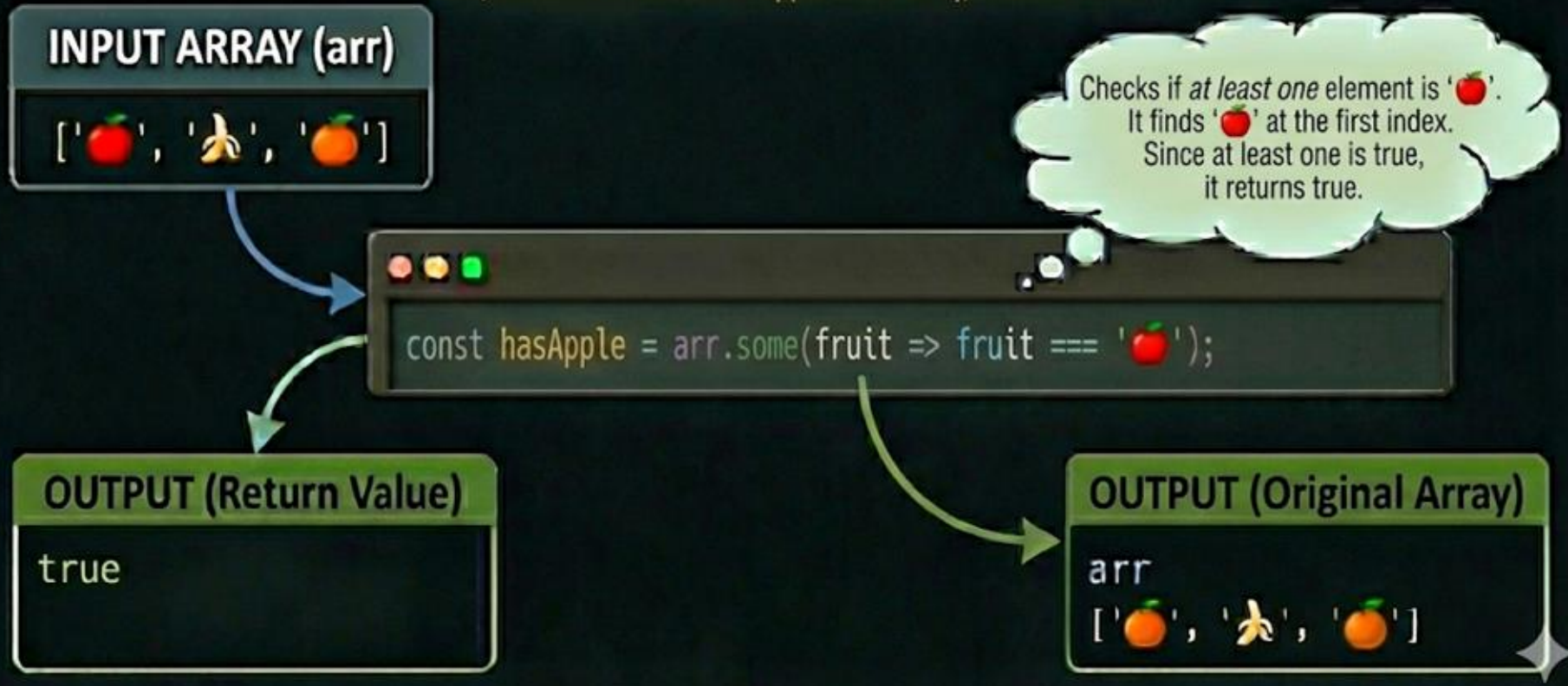
Array.prototype.every() (Non-Mutating)

Tests whether all elements in the array pass the test implemented by the provided function.
Here, we check if every element is a string, which is true.

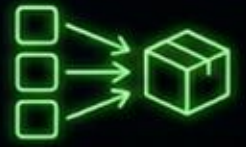


Array.prototype.some() (Non-Mutating)

Tests whether at least one element in the array passes the test implemented by the provided function.
Here, we check if there is an apple in the array, which is true.

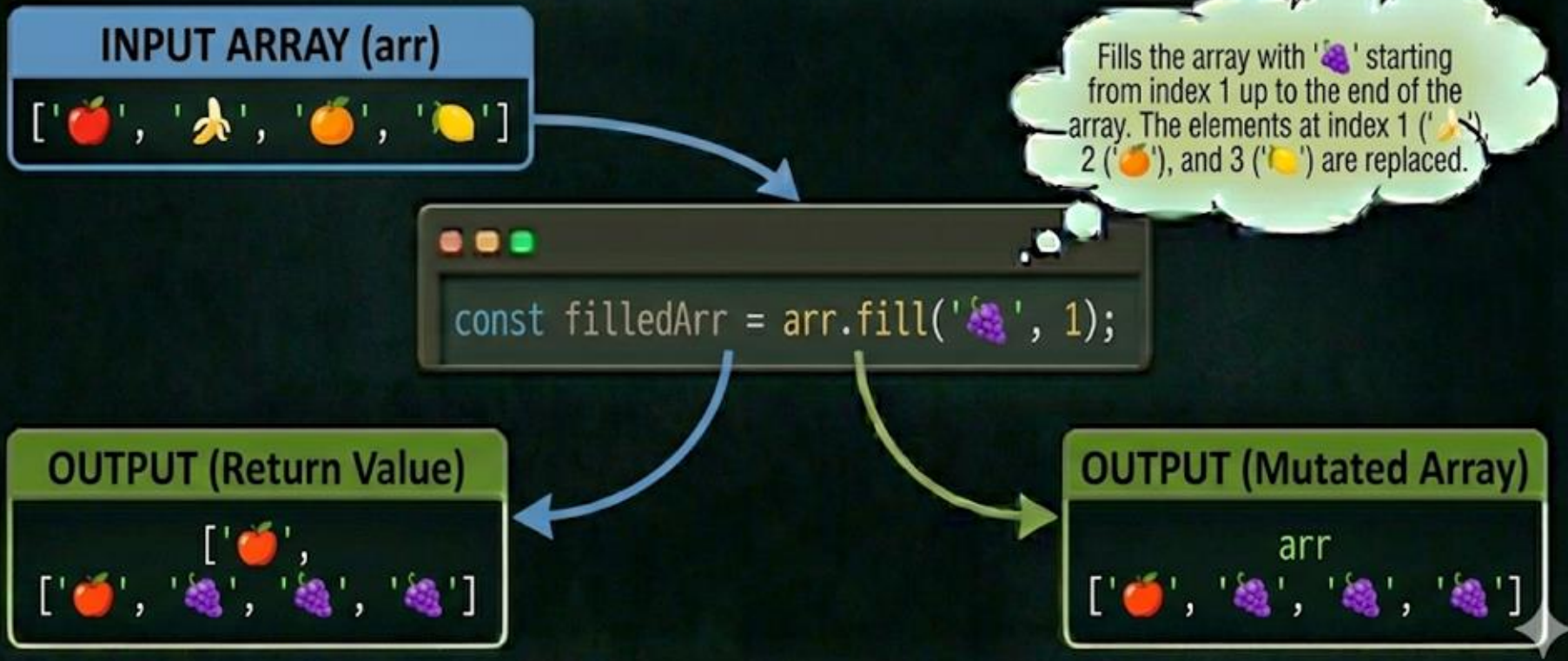


UTILITY & STATIC METHODS

- fill() (Mutating) 
- copyWithin() (Mutating) 
- Array.isArray() (Static) 
- Array.from() (Static) 
- Array.of() (Static) 

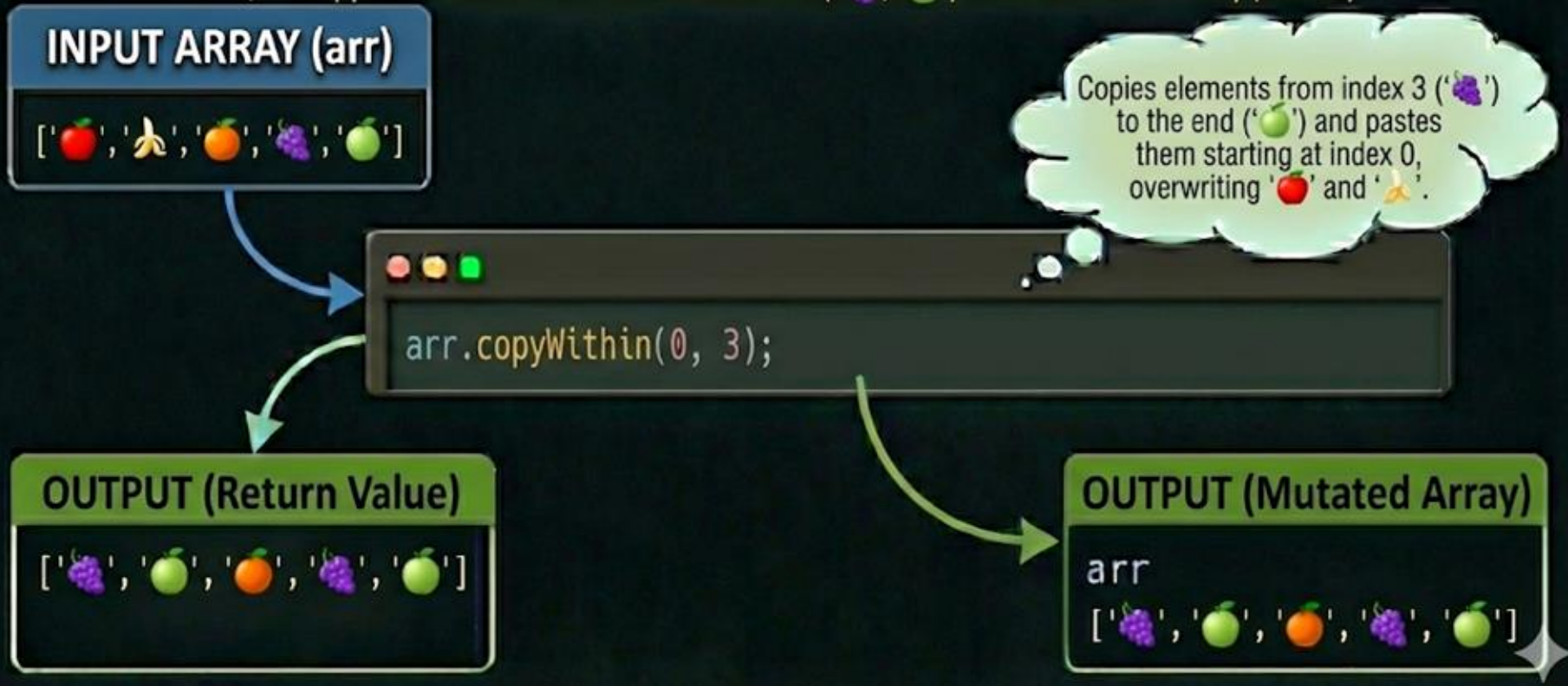
Array.prototype.fill() (Mutating)

Changes all elements in an array to a static value, from a start index (default 0) to an end index (default array.length). Here, we fill the array with grapes from index 1 onwards.



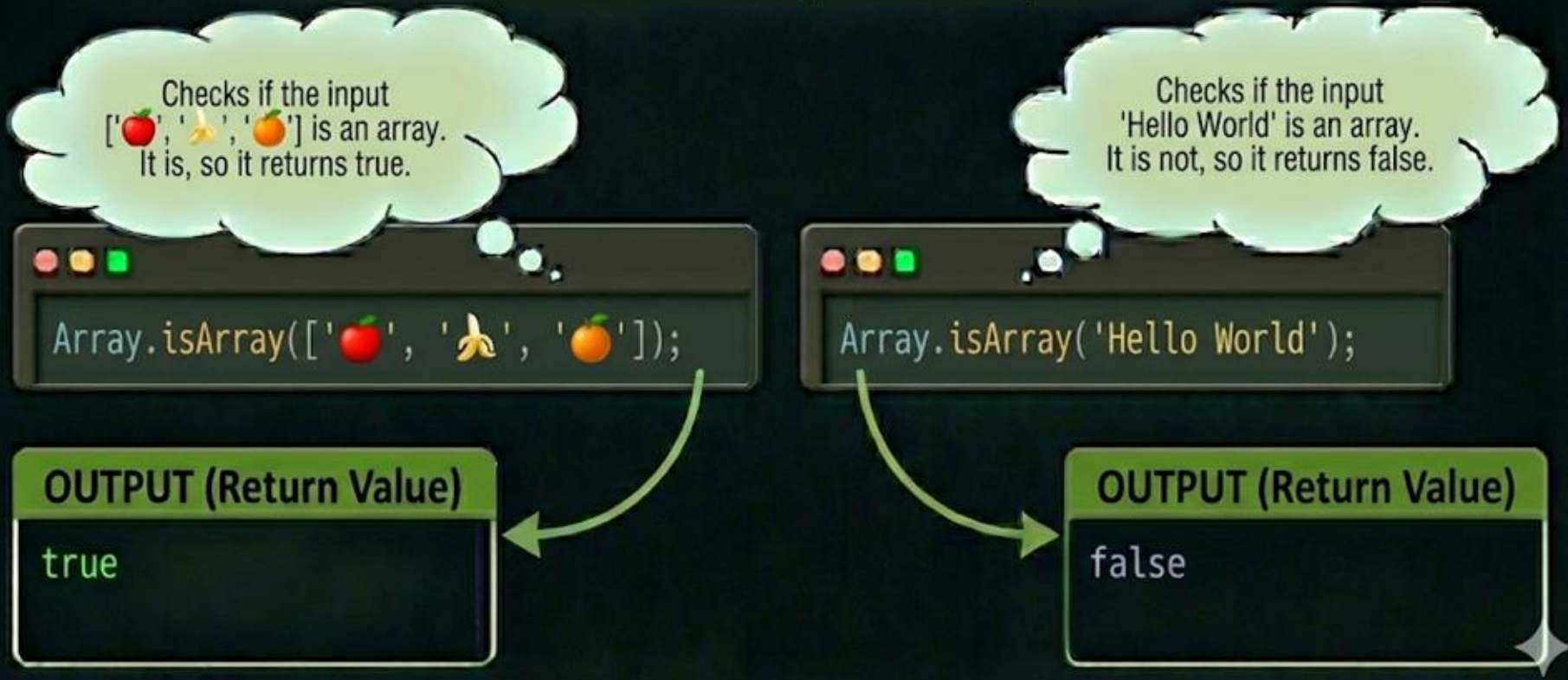
Array.prototype.copyWithin() (Mutating)

Shallow copies part of an array to another location in the same array and returns it without modifying its length.
Here, we copy the elements from index 3 to the end ('🍇', '🍏') to the start of the array (index 0).



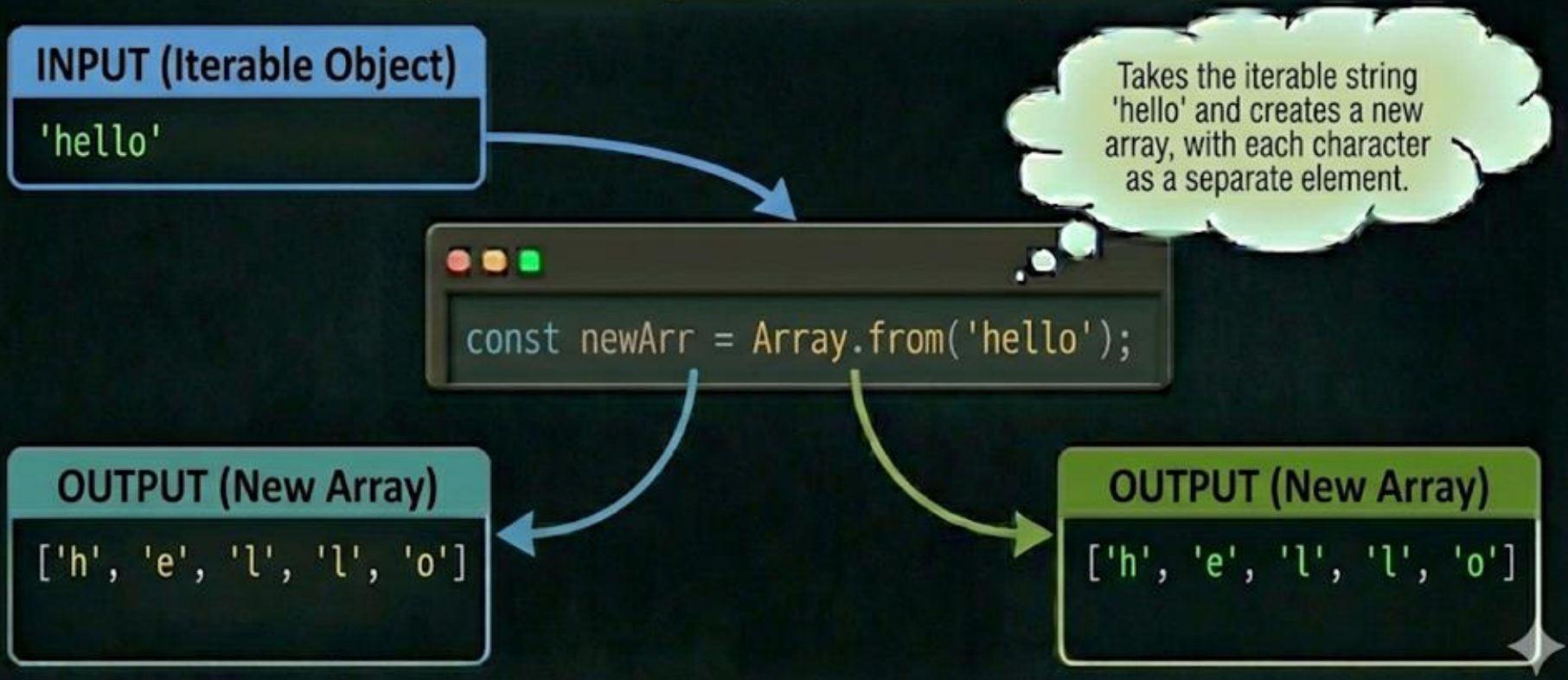
Array.isArray() (Static)

Returns true if the passed value is an Array; otherwise false.
This is a static method, so it's called on the Array constructor itself, not on an array instance.



Array.from() (Static)

Creates a new, shallow-copied Array instance from an array-like or iterable object. Here, we create an array from a string.



Array.of() (Static)

Creates a new Array instance using the variable number of arguments provided as elements.

INPUT (Arguments)

```
1, 'apple', true, null, {id: 1}
```

Takes the provided arguments (a variable number) and creates a new array where each argument becomes an element.

```
const newArr = Array.of(1, 'apple', true, null, {id: 1});
```

OUTPUT (New Array)

```
[1, 'apple', true, null,
{id: 1}]
```