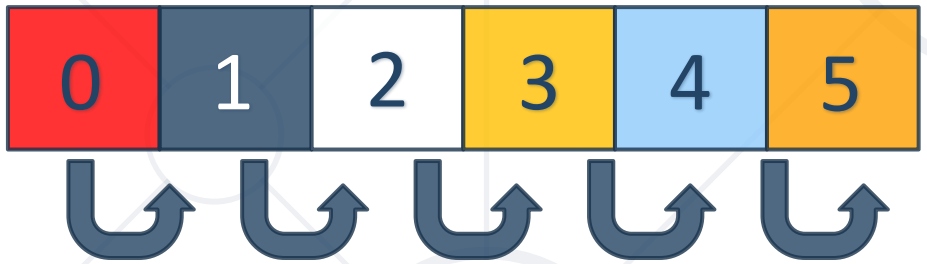
Arrays and Nested Arrays

Definitions and Manipulations



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#js-advanced

Table of Contents

Arrays

What is an Array?

- Arrays are **list-like objects**
- Arrays are a **reference type**, the variable points to an address in memory

- Elements are **numbered** from **0** to **length - 1**
- Creating an array using **an array literal**

```
let numbers = [10, 20, 30, 40, 50];
```

What is an Array?

- Neither the **length** of a JavaScript array **or** the **types** of its elements are **fixed**
- An array's **length can be changed** at any time
- Data can be stored at non-contiguous locations in the array
- JavaScript arrays are not guaranteed to be dense

Arrays of Different Types

```
// Array holding numbers
let numbers = [10, 20, 30, 40, 50];
```

```
// Array holding strings
let weekDays = ['Monday', 'Tuesday', 'Wednesday',
   'Thursday', 'Friday', 'Saturday', 'Sunday'];
```

```
// Array holding mixed data (not a good practice)
let mixedArr = [20, new Date(), 'hello', {x:5, y:8}];
```

Accessing Array Elements

Accessing Elements

Software University

- Array elements are accessed using their **index**

```
let cars = ['BMW', 'Audi', 'Opel'];
let firstCar = cars[0];   // BMW
let lastCar = cars[cars.length - 1];  // Opel
```

- Accessing indexes that do not exist in the array returns **undefined**

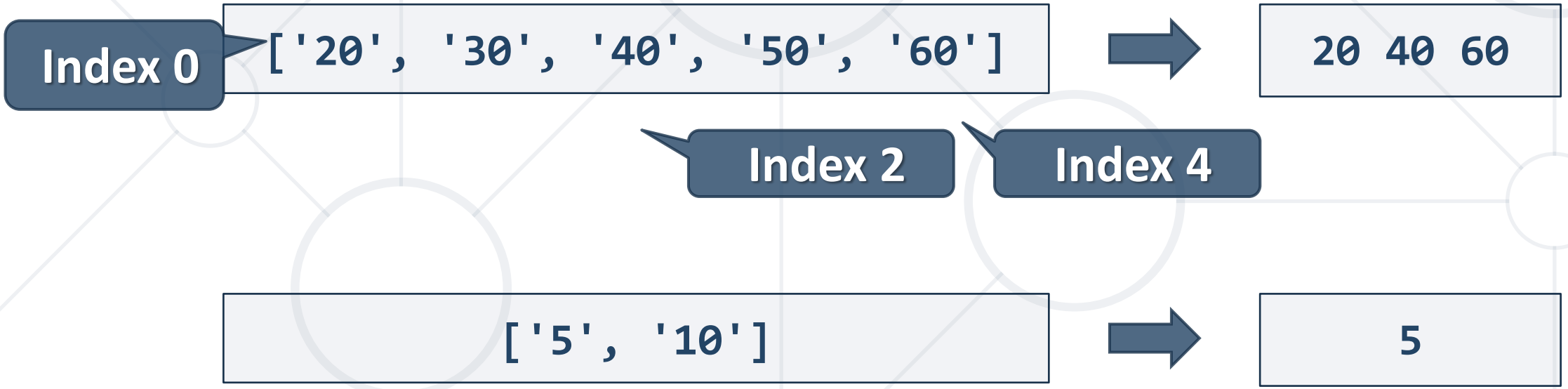
```
console.log(cars[3]);   // undefined
console.log(cars[-1]);  // undefined
```

- Arrays can be **iterated** using **for-of** loop

```
for (let car of cars) { … }
```

Problem: Even Position Element

- Find every element at **even index** in input array
- **Print** them on the console, separated by space



Solution: Even Position Element

```
function solve(arr) {
  let result = '';

  for (let i = 0; i < arr.length; i+=2) {
    result += arr[i];
    result += ' ';
  }

  console.log(result);
}
```

Arrays Indexation

Software University

- Setting values via **non-integers** using **bracket notation** (or dot notation) creates **object properties** instead of array elements (will be discussed in later lesson)

```
let arr = [];
arr[3.4] = 'Oranges';
arr[-1] = 'Apples';
console.log(arr.length);                    // 0
console.log(arr.hasOwnProperty(3.4));       // true

arr["1"] = 'Grapes';
console.log(arr.length);                    // 2
console.log(arr); // [ <1 empty item>, 'Grapes',
'3.4': 'Oranges', '-1': 'Apples' ]
```

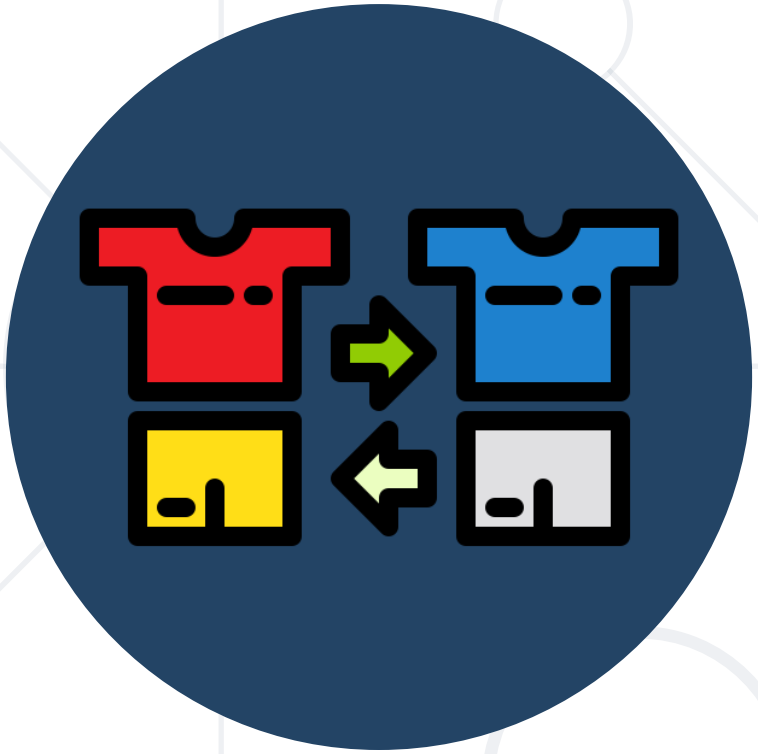
Destructuring Syntax

- Expression that **unpacks values** from **arrays** or **objects**, into distinct **variables**

```
let numbers = [10, 20, 30, 40, 50];
let [a, b, ...elems] = numbers;

console.log(a) // 10
console.log(b) // 20
console.log(elems) // [30, 40, 50]
```

Rest operator

- The **rest operator** can also be used to collect function parameters into an array



Mutator Methods

Modify the Array

Pop

- Removes the **last element** from an array and returns that element
- This method **changes** the **length** of the array

```
let nums = [10, 20, 30, 40, 50, 60, 70];
console.log(nums.length); // 7
console.log(nums.pop());  // 70
console.log(nums.length); // 6
console.log(nums);        // [ 10, 20, 30, 40, 50, 60 ]
```

Push

- The **push()** method **adds one or more** elements to the **end** of an array and **returns** the new **length** of the array

```
let nums = [10, 20, 30, 40, 50, 60, 70];
console.log(nums.length);   // 7
console.log(nums.push(80)); // 8 (nums.length)
console.log(nums); // [ 10, 20, 30, 40, 50, 60, 70, 80 ]
```

Shift

- The `shift()` method **removes** the **first element** from an array and **returns** that **removed element**
- This method **changes** the **length** of the array

```
let nums = [10, 20, 30, 40, 50, 60, 70];
console.log(nums.length); // 7
console.log(nums.shift()); // 10 (removed element)
console.log(nums);  // [ 20, 30, 40, 50, 60, 70 ]
```

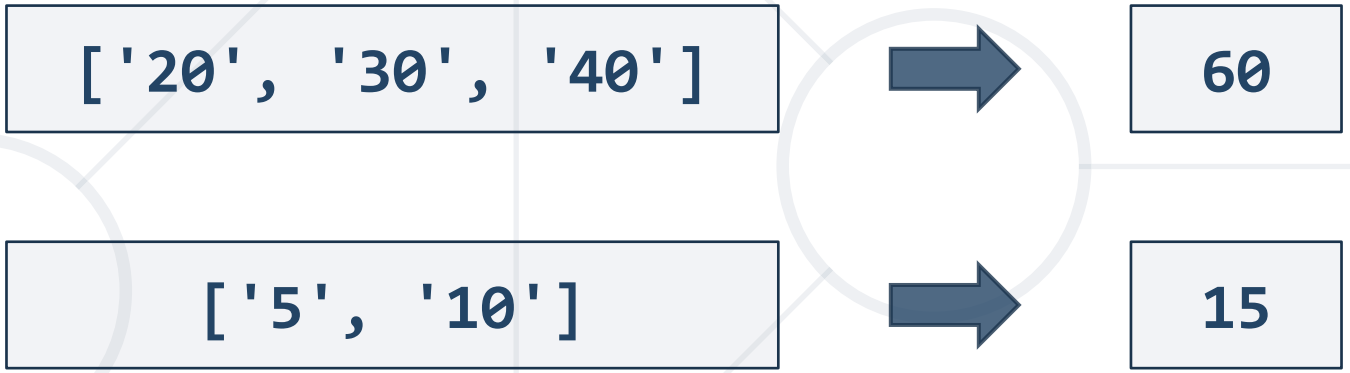
Unshift

- The `unshift()` method **adds one or more** elements to the **beginning** of an array and **returns** the new **length** of the array

```
let nums = [40, 50, 60];
console.log(nums.length);          // 3
console.log(nums.unshift(30));  // 4 (nums.length)
console.log(nums.unshift(10,20)); // 6 (nums.length)
console.log(nums);  // [ 10, 20, 30, 40, 50, 60 ]
```

Problem: Sum First and Last

- Receive an **array of strings** as **input**
- Calculate the **sum** of the **first** and **last** elements
- **Return** the value at the end of your function



Solution: Sum First and Last

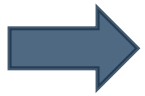
```
function firstSolution(arr) {
  const first = Number(arr[0]);
  const last = Number(arr[arr.length - 1]);

  return first + last;
}
```

```
function secondSolution(arr) {
  return Number(arr.pop()) + Number(arr.shift());
}
```

Problem: Negative / Positive Numbers

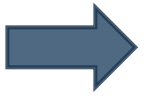
- Create a **new array** from the input array
  - **Prepend** negative elements at the front of the result
  - **Append** non-negative elements at the end of the result
- **Print** each resulting value on a new line

```
[7, -2, 8, 9]
```



```
-2
7
8
9
```

```
[3, -2, 0, -1]
```



```
-1
-2
3
0
```

Solution: Negative / Positive Numbers

Software University

```
function solve(arr) {
  const result =[];

  for (let num of arr) {
    if (num < 0) { result.unshift(num); }
    else { result.push(num); }
  }

  for (let num of result) {
    console.log(num);
  }
}
```

Splice

- Changes the contents of an array by **removing** or **replacing** existing **elements** and/or **adding new** elements

```
let nums = [1, 3, 4, 5, 6];
nums.splice(1, 0, 2); // inserts at index 1
console.log(nums); // [ 1, 2, 3, 4, 5, 6 ]
nums.splice(4, 1, 19); // replaces 1 element at index 4
console.log(nums); // [ 1, 2, 3, 4, 19, 6 ]
let el = nums.splice(2, 1); // removes 1 element at index 2
console.log(nums); // [ 1, 2, 4, 19, 6 ]
console.log(el); // [ 3 ]
```

Fill

- Fills all the elements of an array from a **start index** to an **end index** with a **static value**

```
let arr = [1, 2, 3, 4];
// fill with 0 from position 2 until position 4
console.log(arr.fill(0, 2, 4)); // [1, 2, 0, 0]
// fill with 5 from position 1
console.log(arr.fill(5, 1)); // [1, 5, 5, 5]
console.log(arr.fill(6)); // [6, 6, 6, 6]
```

Reverse

- Reverses the array
  - The **first** array **element becomes** the **last**, and the last array element becomes the first

```
let arr = [1, 2, 3, 4];
arr.reverse();
console.log(arr); // [ 4, 3, 2, 1 ]
```

Sorting Arrays

Sorting Arrays

- The `sort()` method sorts the items of an array
- Depending on the provided **compare function**, sorting can be **alphabetic** or **numeric**,
  and either **ascending (up)** or **descending (down)**
- By default, the `sort()` function sorts the values as strings in **alphabetical** and **ascending** order
- If you want to sort numbers or other values, you need to provide the correct **compare function**!

Sorting Arrays – Example

```
let names = ["Peter","George","Mary"];
names.sort(); // Default behaviour – alphabetical order
console.log(names); // ["George","Mary","Peter"]
```

```
let numbers = [20, 40, 10, 30, 100, 5];
numbers.sort(); // Unexpected result on arrays of numbers!
console.log(numbers); // [10, 100, 20, 30, 40, 5]
```

Compare Functions

- A **function** receiving **two parameters**, e.g. **a** and **b**
  - **Returns** either a **positive** number, a **negative** number, or **zero**
  - If **result < 0**, a is sorted **before** b
  - If **result > 0**, a is sorted **after** b
  - If **result = 0**, a and b are **equal** (no change)

```
let nums = [20, 40, 10, 30, 100, 5];
nums.sort((a, b) => a - b); // Compare elements as numbers
console.log(nums.join('|')); // 5|10|20|30|40|100
```

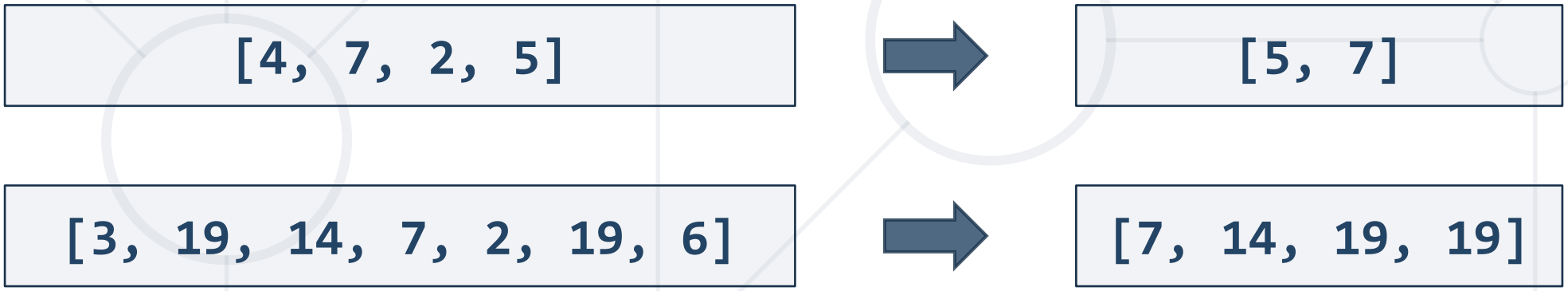
Sorting String Arrays

- The `localeCompare()` method is used to compare any two characters without regard for the case used
  - It's a string method so it can't be used directly on an array
  - Pass `localeCompare()` as the comparison function

```
let words = ['nest', 'Eggs', 'bite', 'Grip', 'jAw'];
words.sort((a, b) => a.localeCompare(b));
// ['bite', 'Eggs', 'Grip', 'jAw', 'nest']
```

Problem: Bigger Half

- **Sort** an input array of **numbers** in **ascending** order
- Create a **new array** from the **second half** of the input array
  - If there are an **odd number** of elements, take the **bigger half**
- **Return** the resulting array



Solution: Bigger Half

```
function solve(arr) {
  arr.sort((a, b) => a - b);
  const middle = Math.floor(arr.length / 2);
  const result = arr.slice(middle);

  return result;
}
```

Accessor Methods

Join

- Creates and returns a **new string** by **concatenating** all of the elements in an array (or an array-like object), **separated** by commas or a **specified separator** string

```
let elements = ['Fire', 'Air', 'Water'];
console.log(elements.join()); // "Fire,Air,Water"
console.log(elements.join('')); // "FireAirWater"
console.log(elements.join('-')); // "Fire-Air-Water"
console.log(['Fire'].join("."); // Fire
```

Concat

- The `concat()` method is used to **merge** two or more arrays
- This method **does not change** the **existing arrays**, but instead returns a new array

```
const num1 = [1, 2, 3];
const num2 = [4, 5, 6];
const num3 = [7, 8, 9];
const numbers = num1.concat(num2, num3);
console.log(numbers); //  [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Slice

- The `slice()` method **returns** a shallow **copy** of a **portion** of an array into a **new array** object selected from begin to end (end not included)
- The **original array** will **not** be **modified**

```
let fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];
let citrus = fruits.slice(1, 3);
let fruitsCopy = fruits.slice();
// fruits contains ['Banana', 'Orange', 'Lemon', 'Apple',
'Mango']
// citrus contains ['Orange', 'Lemon']
```

Includes

- Determines whether an array contains a certain element, returning **true** or **false** as appropriate

```
// array length is 3
// fromIndex is -100
// computed index is 3 + (-100) = -97
let arr = ['a', 'b', 'c'];
arr.includes('a', -100); // true
arr.includes('b', -100); // true
arr.includes('c', -100); // true
arr.includes('a', -2); // false
```

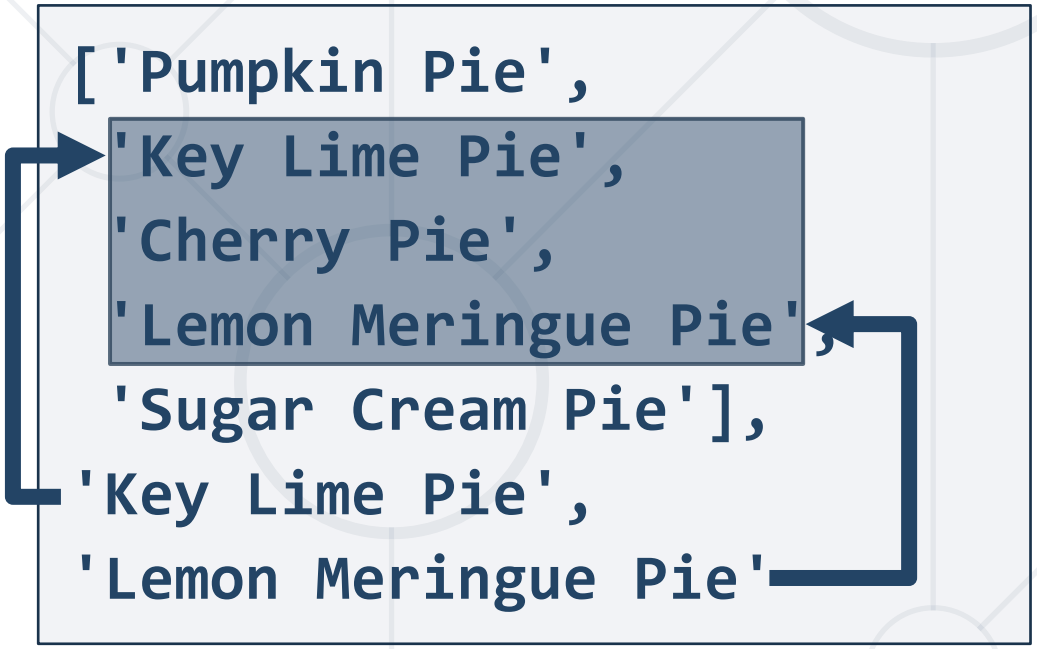
IndexOf

- The `indexOf()` method **returns** the **first index** at which a given **element** can be **found** in the array
  - Output is **-1** if element is **not present**

```
const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison')); // 1
// start from index 2
console.log(beasts.indexOf('bison', 2)); // 4
console.log(beasts.indexOf('giraffe')); // -1
```

Problem: Piece of Pie

- Receive **three parameters** – an **array of pies** and two **strings**
- Take all pie flavors **between** and **including** the two strings
- **Return** the result as an **array of strings**




```
['Key Lime Pie',
 'Cherry Pie',
 'Lemon Meringue Pie']
```

Solution: Piece of Pie

```
function solve(pies, startFlavor, endFlavor) {
  const start = pies.indexOf(startFlavor);
  const end = pies.indexOf(endFlavor) + 1;

  const result = pies.slice(start, end);

  return result;
}
```

Iteration Methods

ForEach

- The `forEach()` method **executes a provided function** once for each array element
- Converting a for loop to forEach

```
const items = ['item1', 'item2', 'item3'];
const copy = [];

// For Loop
for (let i = 0; i < items.length; i++) {
  copy.push(items[i]);
}

// ForEach
items.forEach(item => { copy.push(item); });
```

Map

- **Creates a new array** with the results of calling a **provided function** on every element in the calling array

```
let numbers = [1, 4, 9];
let roots = numbers.map(function(num, i, arr) {
  return Math.sqrt(num)
});
// roots is now [1, 2, 3]
// numbers is still [1, 4, 9]
```

Some

- The `some()` method **tests** whether **at least one** element in the array passes the test implemented by the **provided function**
- It returns a **Boolean** value

```
let array = [1, 2, 3, 4, 5];
let isEven = function(element) {
  // checks whether an element is even
  return element % 2 === 0;
};
console.log(array.some(isEven)); // true
```

Find

- Returns the **first found value** in the array, if an **element** in the array **satisfies** the **provided** testing **function** or **undefined** if not found

```
let array1 = [5, 12, 8, 130, 44];
let found = array1.find(function(element) {
  return element > 10;
});
console.log(found); // 12
```

Filter

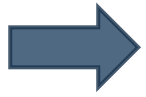
- Creates a **new array** with **filtered elements only**
- Calls a **provided** callback **function** once for each element in an array
- **Does not mutate** the **array** on which it is called

```
let fruits = ['apple', 'banana', 'grapes', 'mango', 'orange'];
 // Filter array items based on search criteria (query)
function filterItems(arr, query) {
  return arr.filter(function(el) {
      return el.toLowerCase().indexOf(query.toLowerCase()) !== -1;
  });
};
console.log(filterItems(fruits, 'ap')); // ['apple', 'grapes']
```

Problem: Process Odd Positions

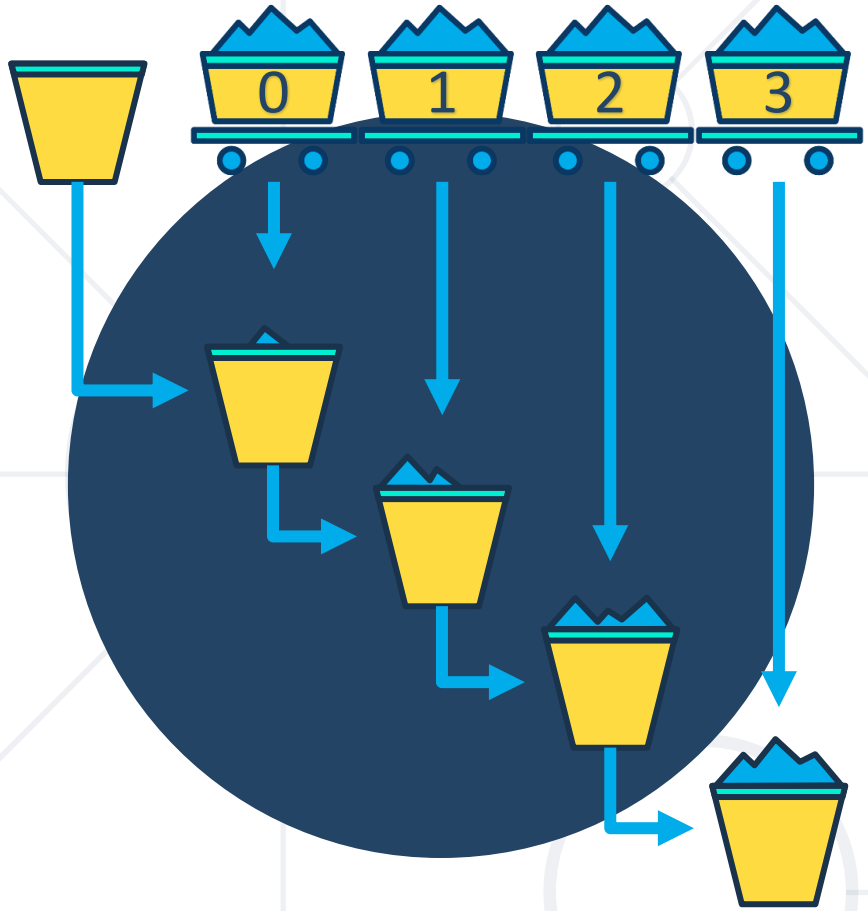
You are given **array of numbers**

- Find all elements at **odd** positions (indexes)
- **Multiply** them by 2
- **Reverse** them
- Return the elements separated with a single space

`[10, 15, 20, 25]`		`50 30`
`[3, 0, 10, 4, 7, 3]`		`6 8 0`

Solution: Process Odd Positions

```
function solve(arr) {
  return arr.filter((a, i) => i % 2 !== 0)
    .map(x => x * 2)
    .reverse()
    .join(' ');
}
```



Reducing Arrays

Reduce

- The `reduce()` method executes a reducer function on each element of the array, resulting in a **single output value**

```
const array1 = [1, 2, 3, 4];
const reducer =
(accumulator, currentValue) => accumulator + currentValue;
console.log(array1.reduce(reducer)); // 10
console.log(array1.reduce(reducer, 5)); // 15
```

Reducer Function

- The reducer function takes **four** arguments:
  - Accumulator
  - Current Value
  - Current Index (Optional)
  - Source Array (Optional)
- Your **reducer function's** returned value is **assigned** to the **accumulator**
- **Accumulator's value** - the **final**, **single** resulting **value**

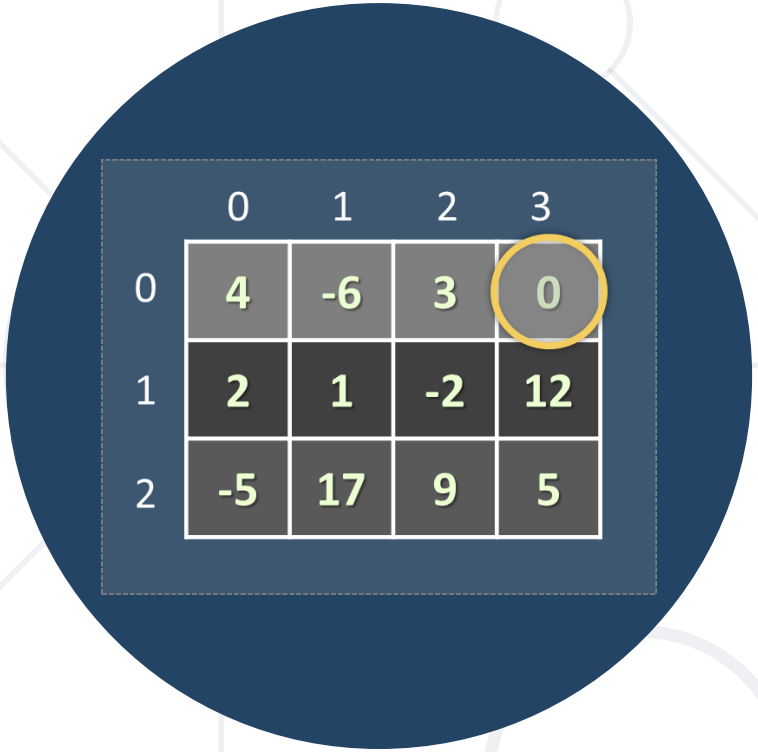
Examples

- Sum all values

```
let sum = [0, 1, 2, 3].reduce(function (acc, curr) {
      return acc + curr;
    }, 0);
console.log(sum); // 6
```

- Finding an average with reduce

```
const numbersArr= [30, 50, 40, 10, 70];
const average =
  numbersArr.reduce((total, number, index, array) => {
        total += number;
        if( index === array.length-1) {
            return total/array.length;
        } else {  return total; }
  });
console.log(average) // 40
```



Nested Arrays

Nested Arrays in JS

Array of 4 arrays

Element arr[2][0] at row 2, column 0

	0	1	2	3
0	4	6	3	0
1	2	1	-2	
2	-5	17		
3	7	3	9	12

```
let arr = [
    [4, 6, 3, 0],
    [2, 1, -2],
    [-5, 17],
    [7, 3, 9, 12]
];
```

Looping Through a Nested Array

Software University

```
let arr = [[4, 5, 6],
          [6, 5, 4],
          [5, 5, 5]];
```

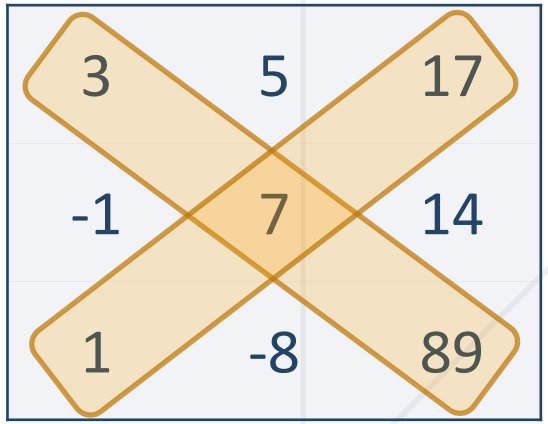
```
arr.forEach(printRow);
function printRow(row){
    console.log(row);
    row.forEach(printNumber);
}
function printNumber(num){
    console.log(num);
};
```

Prints each row of the array on a separate line

Prints each element of the array on a separate line

Problem: Diagonal Sums

- You are given an **array of arrays**, containing number elements
  - Find what is the **sum** at the **main** diagonal
  - Find what is the **sum** at the **secondary** diagonal
  - Print the diagonal sums separated by **space**



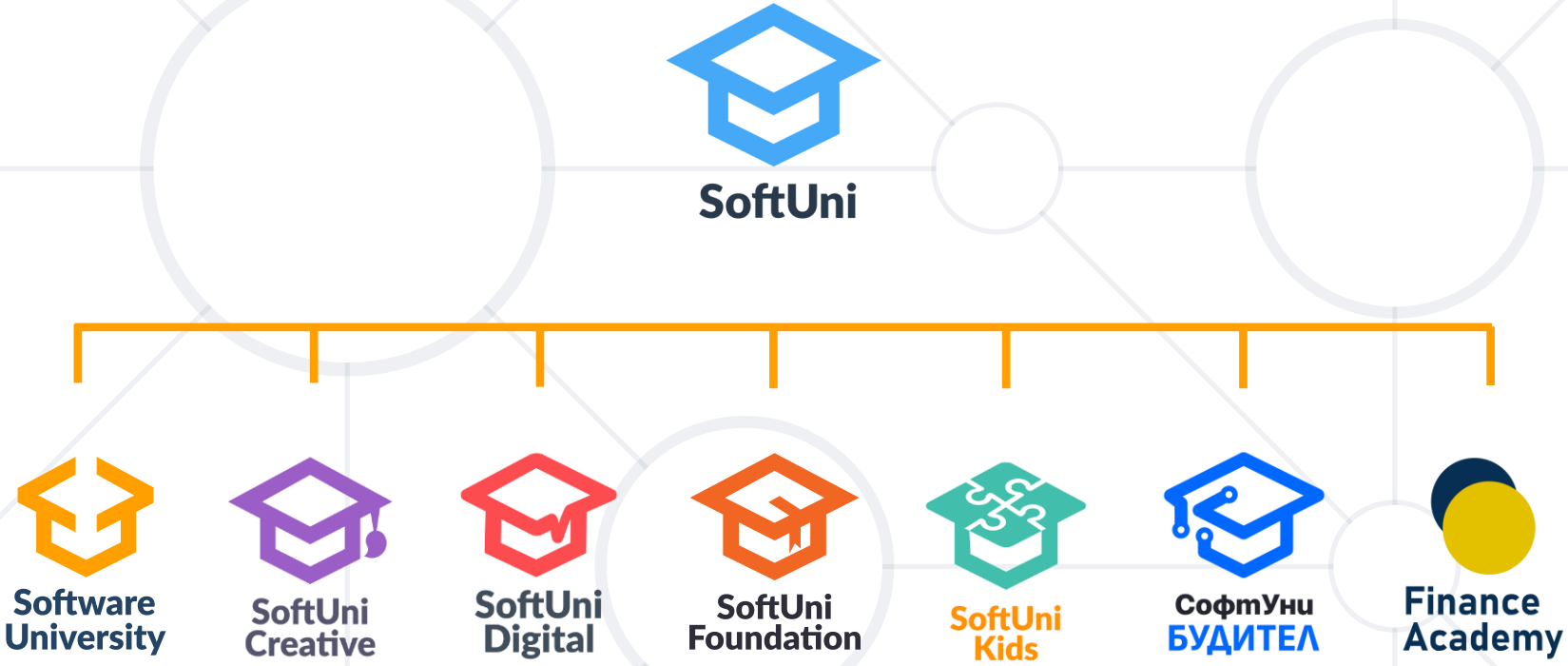
Solution: Diagonal Sums

```
function diagonalSums(input) {
    let firstDiagonal = 0;
    let secondDiagonal = 0;
    let firstIndex = 0;
    let secondIndex = input[0].length - 1;
    input.forEach(array => {
        firstDiagonal += array[firstIndex++];
        secondDiagonal += array[secondIndex--];

    });
    console.log(firstDiagonal + ' ' + secondDiagonal);
}
```

Summary

- Arrays are list-like objects
- Elements are accessed using their index
- Mutator methods change the original array
- Accessor methods return a new array
- Arrays can be reduced to a single value
- An array of arrays is called a matrix
- Matrices can have more than 2 dimensions

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg