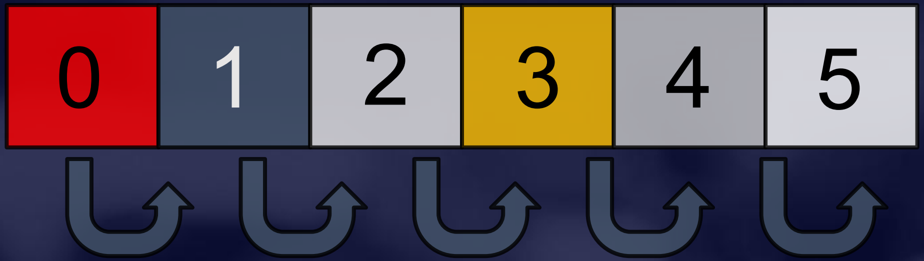
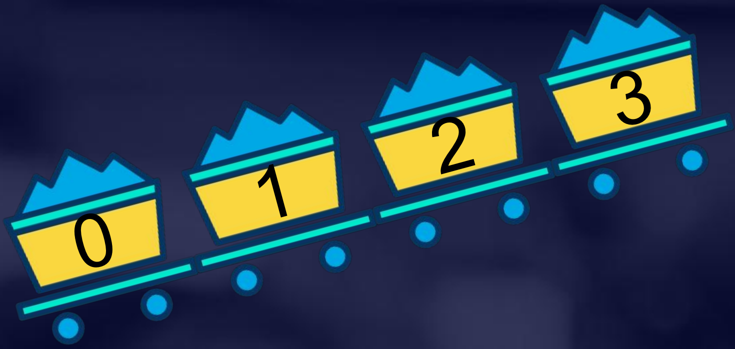
KINGSLAND UNIVERSITY

Arrays and Nested Arrays



Definitions and Manipulations

Table of Contents

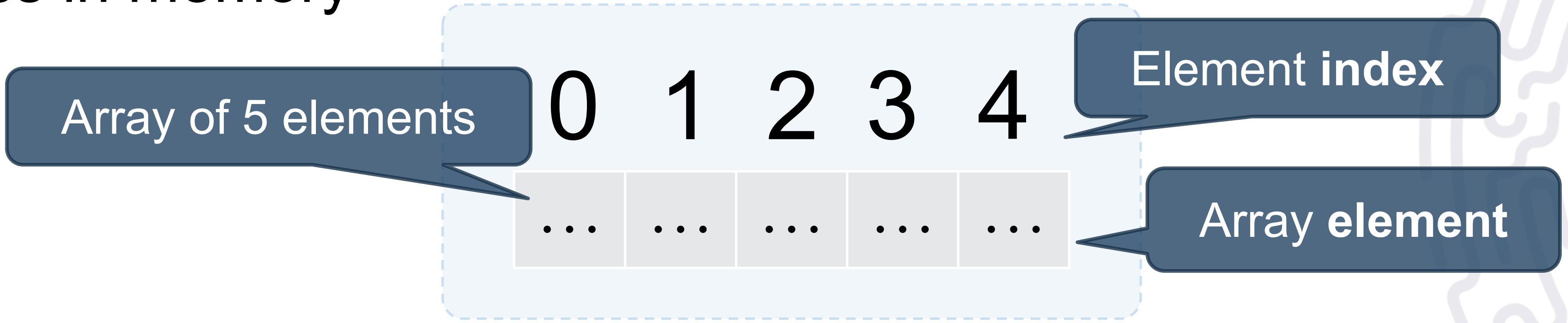


Working with Arrays of Elements

Arrays in JS

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 **nor** the **types** of its elements are **fixed**
- An array's **length can change** 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}];
```

Arrays Indexation

- Setting or accessing via non-integers using **bracket notation** (or dot notation) will **not** set or retrieve an element from the **array** list **itself**
  - It will set or access a **variable** associated with that **array's** object **property collection**
- The array's object **properties** and list of array **elements** are **separate**

Examples

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

```
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' ]
```

Accessing Array Elements

Accessing Elements

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

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

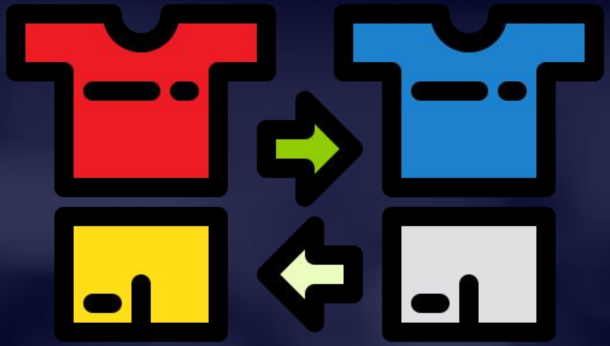
- Accessing indexes that do not exist in the array returns **undefined**

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

Accessing Elements

- Array elements are **object properties**
- Trying to access an element of an array as follows throws a **syntax error** because the property name is **not valid**

```
let years = [1950, 1960, 1970, 1980, 1990, 2000];
console.log(years.0);    // a syntax error
console.log(years[0]);   // works properly
```



Modify the Array

Mutator Methods

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 ]
```

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 ]
```

Sort

- The **`sort()`** method **sorts** the elements of an array in place and returns the sorted array
- The **default sort order** is built upon **converting** the elements into strings, **then comparing** their sequences of UTF-16 code units values
- The **time** and **space complexity** of the sort cannot be guaranteed
  - It depends on the **implementation**

Sort Examples

```
let months = ['March', 'Jan', 'Feb', 'Dec'];

months.sort();

console.log(months); // ["Dec", "Feb", "Jan", "March"]
```

```
let array1 = [1, 30, 4, 21, 100000];

array1.sort();

console.log(array1); // [1, 100000, 21, 30, 4]
```

```
let array2 = [1, 30, 4, 21, 100000];

array2.sort(compareNumbers);

console.log(array2); // [ 1, 4, 21, 30, 100000 ]

function compareNumbers(a, b) { return a - b; }
```

Sorting Objects

- Objects can be sorted, given the **value** of one of their **properties**

```
let items = [
    { name: 'Edward', value: 21 },
    { name: 'Sharpe', value: 37 },
    { name: 'And', value: 45 }
];
// sort by value
items.sort(function (a, b) {
    return a.value - b.value;
});
// sort by name
items.sort(function (a, b) {
    let nameA = a.name.toUpperCase(); // ignore upper and lowercase
    let nameB = b.name.toUpperCase(); // ignore upper and lowercase
    if (nameA < nameB) { return -1; }
    if (nameA > nameB) { return 1; }
    return 0;
});
```

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
```

IndexOf

- The **`indexOf()`** method **returns** the **first index** at which a given **element** can be **found** in the array, or **-1** if it 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
```

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]
```

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
```

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']
```

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); });
```

Filter

- Creates a **new array** with **all elements that pass** the test implemented by the provided function
- Calls a **provided** callback **function** once for each element in an array
- Constructs a **new array** of all the values for which callback returns a value that coerces to **true**
- **Does not mutate** the **array** on which it is called

Filter Example

```
function isBigEnough(value) {
   return value >= 10;
};
let filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

```
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']
```

Find

- Returns the **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
```

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 even = function(element) {
  // checks whether an element is even
  return element % 2 === 0;
};
console.log(array.some(even)); //true
```

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

Map

- Reformatting an Array of Objects

```
const myUsers = [
    { name: 'chuloo', likes: 'grilled chicken' },
    { name: 'chris', likes: 'cold beer' },
    { name: 'sam', likes: 'fish biscuits' }
];
const usersByFood = myUsers.map(item => {
    const container = {};
    container[item.name] = item.likes;
    container.age = item.name.length * 10;
    return container;
});
console.log(usersByFood);
```

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
```

- The reduce method accepts **2 parameters**
  - Reducer function
  - Initial value

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

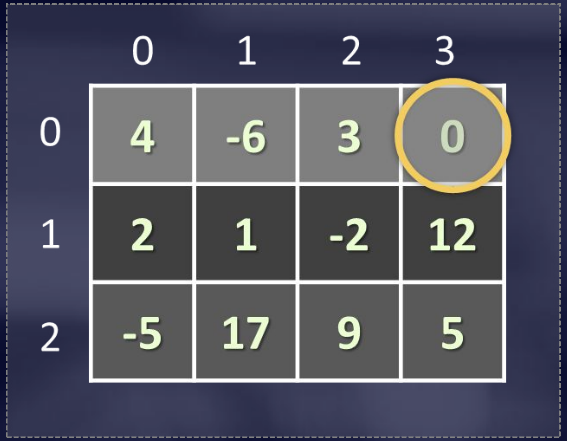
```
let sum = [0, 1, 2, 3].reduce(function (acc, curr) {
     return acc + curr;
   }, 0);
console.log(sum); // 6
```

- Sum of values in an object array - you must supply an initial value

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

Problem: Process Odd Numbers

- You are given **array of numbers**
  - Find all elements in **odd** position
  - **Multiply** them by 2
  - **Reverse** them
  - Print the elements separated with a single space

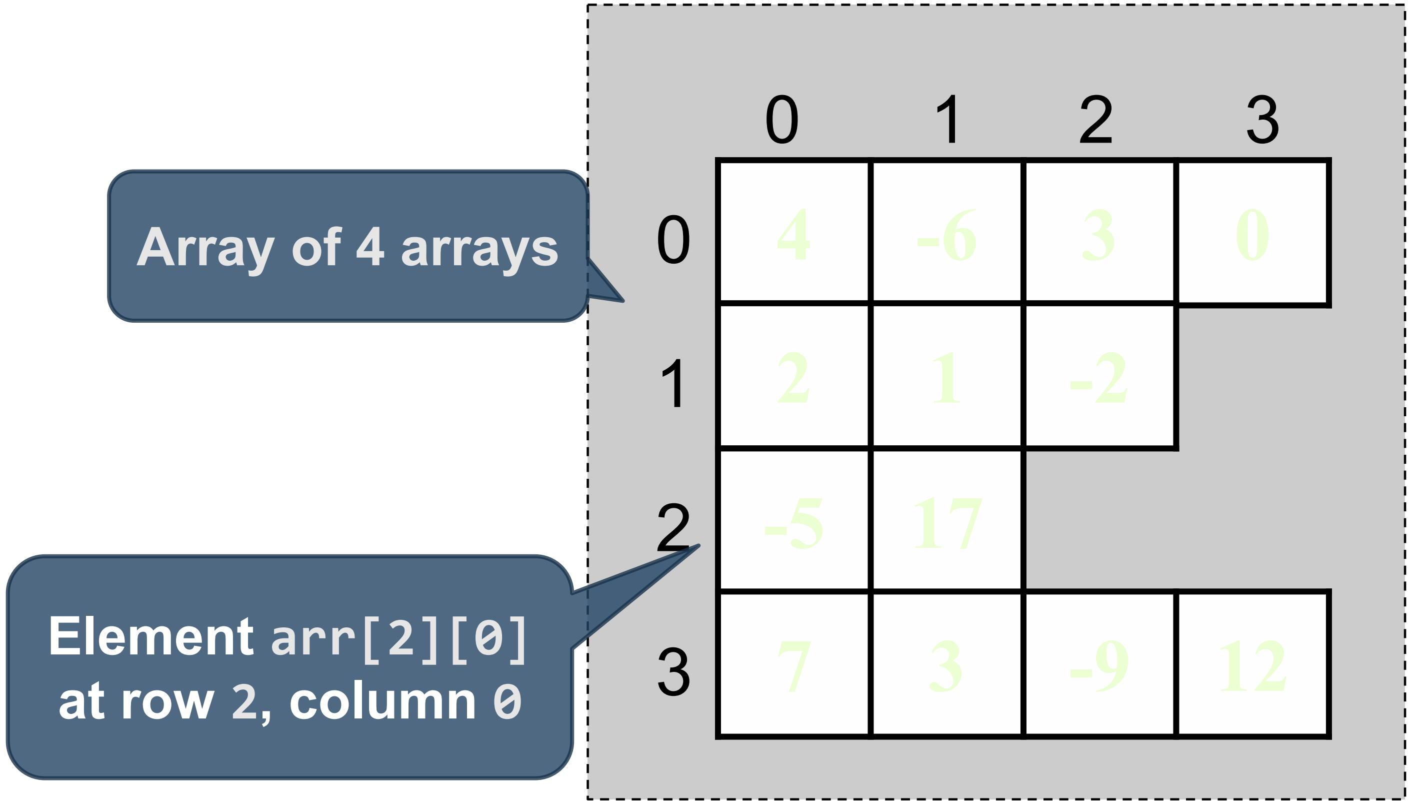


	0	1	2	3
0	4	-6	3	0
1	2	1	-2	12
2	-5	17	9	5

Array of Arrays

Nested Arrays

Nested Arrays in JS



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 4 & -6 & 3 & 0 \\ 1 & 2 & 1 & -2 & \\ 2 & -5 & 17 & & \\ 3 & 7 & 3 & -9 & 12 \end{array}$$

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

Looping Through a Nested Array

```
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

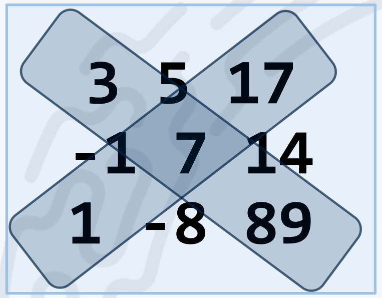
Problem: Diagonal Sums

- 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);
}
```



Live Exercises

Summary

- Arrays are **list-like objects**
- Elements are **accessed** using their **index number**
- **Mutator** methods - methods that **change** the original **array**
- **Accessor** methods - methods that return **new array**
- Looping through arrays
- Nested arrays

Questions?

License

KINGSLAND UNIVERSITY

THANK YOU