Bano Qabil 2.0

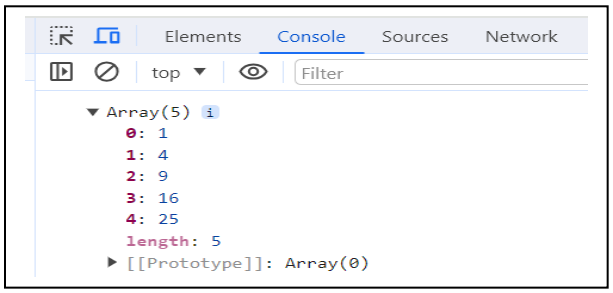
Course: Web – 3 (8 – 10) M

Assignment # 07

Question # 01: Given an array of integers, use the map method to square each element and return a new array with the squared values.

Answer:

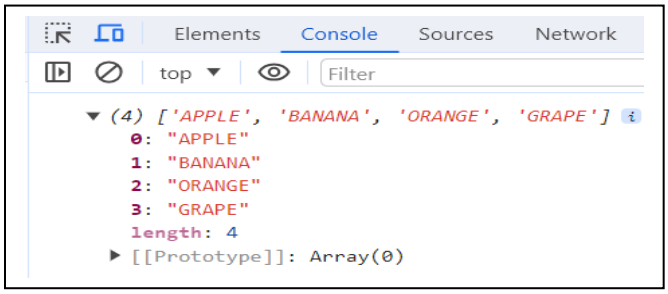
```
const originalArray = [1, 2, 3, 4, 5];
const squaredArray = originalArray.map((element) => element * element);
console.log(squaredArray);
```



Question # 02: Take an array of strings, filter out the ones with a length less than 5, and then capitalize the remaining strings using the map method.

Answer:

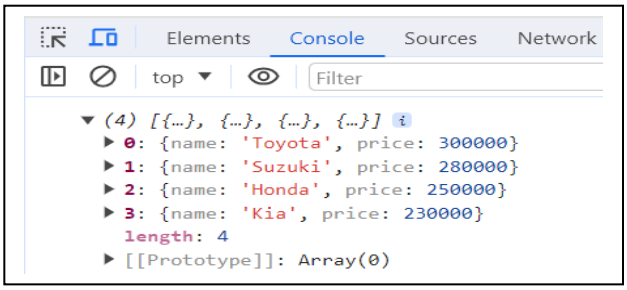
```
const arrayOfStrings = ["apple", "banana", "orange", "kiwi", "grape"];
const filteredAndCapitalized = arrayOfStrings
  .filter((str) => str.length >= 5)
  .map((str) => str.toUpperCase());
console.log(filteredAndCapitalized);
```



Question # 03: Given an array of objects with a 'price' property, use the sort method to arrange them in descending order based on their prices.

Answer:

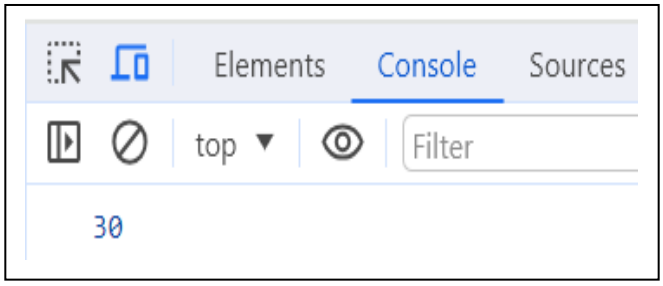
```
const arrayOfObjects = [
    { name: 'Toyota', price: 300000 },
    { name: 'Honda', price: 250000 },
    { name: 'Kia', price: 230000 },
    { name: 'Suzuki', price: 280000 }
  ];
  const sortedArray = arrayOfObjects.sort((a, b) => b.price - a.price);
  console.log(sortedArray);
```



Question # 04: Use the reduce method to find the total sum of all even numbers in an array of integers.

Answer:

```
const arrayOfIntegers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const sumOfEvenNumbers = arrayOfIntegers.reduce((accumulator, currentValue) => {
  if (currentValue % 2 === 0) {
    return accumulator + currentValue;
  }
  return accumulator;
}, 0);
console.log(sumOfEvenNumbers);
```

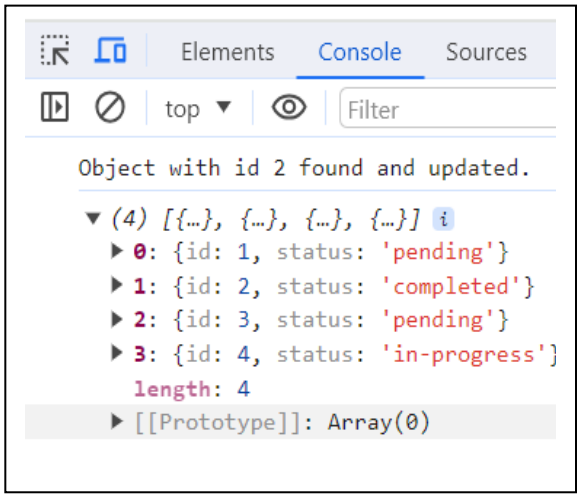


Question # 05: Given an array of objects with 'id' properties, use the find method to locate an object with a specific 'id' and update its 'status' property to 'completed'.

Answer:

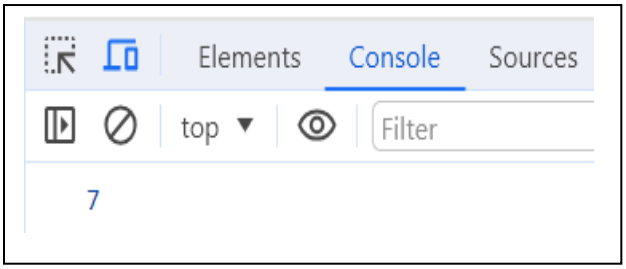
```
const arrayOfObjects = [
    { id: 1, status: 'pending' },
    { id: 2, status: 'in-progress' },
    { id: 3, status: 'pending' },
    { id: 4, status: 'in-progress' }
  ];
  const specificId = 2;
  const foundObject = arrayOfObjects.find(obj => obj.id === specificId);

  if (foundObject) {
    foundObject.status = 'completed';
    console.log(`Object with id ${specificId} found and updated.`);
  } else {
    console.log(`Object with id ${specificId} not found.`);
  }
  console.log(arrayOfObjects);
```



Question # 06: Create a chain of array methods to find the average of all positive numbers in an array of mixed integers and return the result rounded to two decimal places.

```
const mixedIntegers = [3, -1, 7, -4, 12, 0, 8, -2, 5];
const averageOfPositiveNumbers = mixedIntegers
  .filter(number => number > 0)
  .reduce((sum, number, index, array) => {
    sum += number;
    if (index === array.length - 1) {
      return sum / array.length;
    }
    return sum;
  }, 0);
const roundedAverage = averageOfPositiveNumbers.toFixed(2)
console.log(parseFloat(roundedAverage));
```

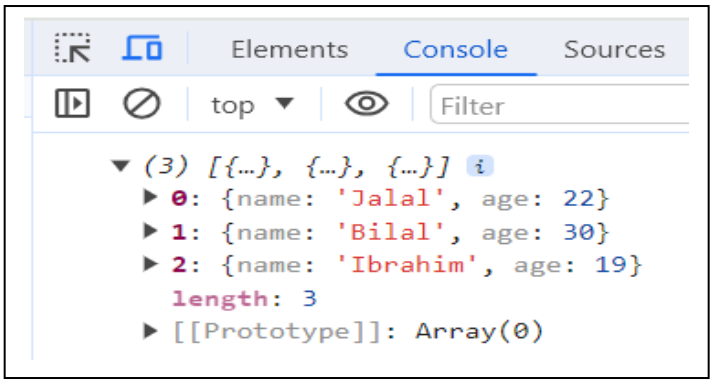


Question # 07: Implement a function that takes an array of objects with 'age' properties and returns an array of those who are adults (age 18 and above) using the filter method.

Answer:

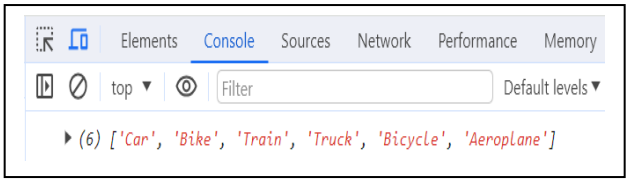
```
function getAdults(persons) {
    return persons.filter(person => person.age >= 18);
  }
  const arrayOfPersons = [
    { name: 'Jalal', age: 22 },
    { name: 'Ali', age: 16 },
    { name: 'Bilal', age: 30 },
    { name: 'Ibrahim', age: 19 }
  ];
  const adults = getAdults(arrayOfPersons);

  console.log(adults);
```



Question # 08: Sort an array of strings based on their lengths in ascending order. If two strings have the same length, maintain their relative order in the sorted array.

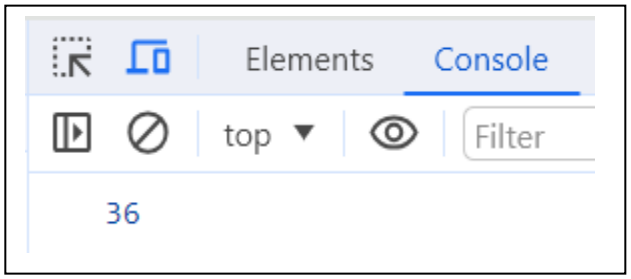
```
const arrayOfStrings = ["Train", "Aeroplane", "Bicycle", "Truck", "Car", "Bike"];
const sortedArray = arrayOfStrings.sort((a, b) => a.length - b.length);
console.log(sortedArray);
```



Question # 09: Given an array of arrays containing numbers, use a combination of array methods to flatten the structure and then calculate the sum of all the numbers.

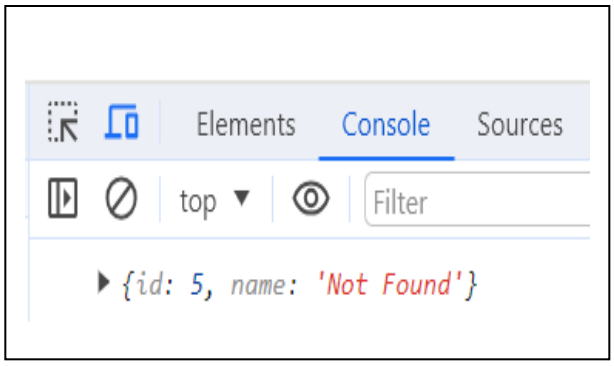
Answer:

```
const arrayOfArrays = [[1, 2, 3], [4, 5], [6, 7, 8]];
const flattenedArray = arrayOfArrays.flat();
const sumOfNumbers = flattenedArray.reduce((sum, number) => sum + number, 0);
console.log(sumOfNumbers);
```



Question # 10: Modify the find method to handle the scenario where the desired element is not found, returning a custom default object instead.

```
const arrayOfObjects = [
    { id: 1, name: 'Jalal' },
    { id: 2, name: 'Ali' },
    { id: 3, name: 'Babar' },
    { id: 4, name: 'Uzair' }
];
const desiredId = 5;
const defaultObject = { id: -1, name: 'Not Found' };
const foundObject = arrayOfObjects.find(obj => obj.id === desiredId) || defaultObject;
console.log(foundObject);
```



Question # 11: How does the map method work in JavaScript, and can you provide an example of when you might use it to manipulate an array of objects?

ANSWER: The **map** method in JavaScript is used to create a new array by applying a given function to each element of the original array. It doesn't modify the original array but instead returns a new array with the results of applying the provided function to each element.

Question # 12: Explain the purpose of the filter method. Provide an example where you use filter to extract elements from an array based on a specific condition.

ANSWER: The filter method in JavaScript is used to create a new array containing only the elements that satisfy a specified condition. It doesn't modify the original array but instead returns a new array with the elements that pass the condition.

Question # 13: Discuss the default behavior of the sort method for strings and numbers. How would you use a custom comparison function to sort an array of objects by a specific property?

ANSWER: The sort method in JavaScript is used to sort the elements of an array. By default, it sorts elements as strings. The default behavior for sorting strings is lexicographic (dictionary) order, and for numbers, it performs a simple numerical sort.

Question # 14: Describe the purpose of the reduce method and provide an example where you use it to compute a single value from an array of numbers.

ANSWER: The reduce method in JavaScript is used to iterate over an array and accumulate a single value based on the elements of the array. It takes a callback function and an optional initial value as parameters.

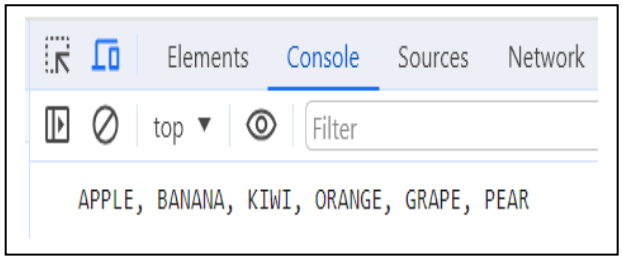
Question # 15: How does the find method differ from filter? Give an example of a scenario where using find is more appropriate than filter.

ANSWER: The find and filter methods in JavaScript are both array methods used for working with arrays and extracting elements based on certain criteria, but they differ in their purpose and behavior.

Question # 16: Create a chain of array methods (map, filter, reduce, etc.) to transform an array of strings into a single concatenated string with a specific condition.

ANSWER:

```
const arrayOfStrings = ["apple", "banana", "kiwi", "orange", "grape", "pear"];
const resultString = arrayOfStrings
  .filter(str => str.length > 3).map(str => str.toUpperCase()).join(', ');
console.log(resultString);
```

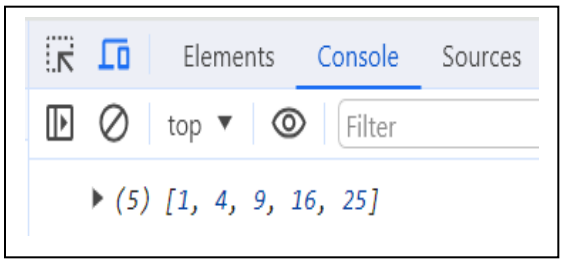


Question # 17: Explain the concept of callback functions in the context of array methods. Provide an example of using a callback function with the map method.

ANSWER: A callback function is a function that is passed as an argument to another function. The purpose of the callback function is to be executed later, typically during the execution of the higher-order function that receives it. Array methods often take callback functions to define the logic that should be applied to each element of the array.

EXAMPLE:

```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(number => number * number);
console.log(squaredNumbers);
```



Question # 18: How would you handle potential errors when using array methods like find or reduce? Provide an example of error handling in such a scenario.

ANSWER: When using array methods like find or reduce, error handling can be implemented by checking the return value and handling unexpected conditions appropriately. These methods typically return a special value (like undefined for find or an accumulator for reduce) when the expected element or result is not found.

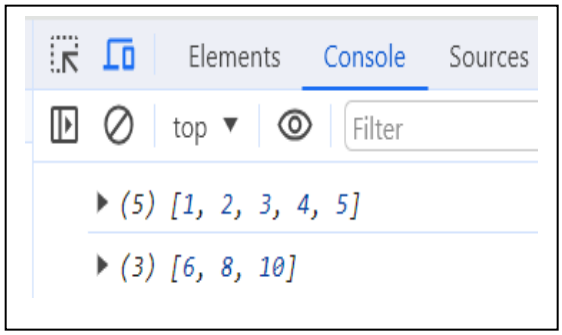
Question # 19: Discuss the importance of immutability when working with array methods. Demonstrate how you would perform immutable operations using methods like map or filter.

ANSWER: Immutability is an important concept in programming that refers to the state of an object not being modified after it's created. In the context of working with arrays and array methods in JavaScript, embracing immutability can lead to more predictable and maintainable code. Instead of

modifying the existing array, you create a new array with the desired changes. This helps avoid unintended side effects and makes it easier to reason about the behavior of our code.

EXAMPLE:

```
const originalArray = [1, 2, 3, 4, 5];
const transformedArray = originalArray
  .map(number => number * 2)
  .filter(number => number > 4);
console.log(originalArray);
console.log(transformedArray);
```



Question # 20: Compare the performance implications of using map versus forEach. In what scenarios would you prefer one over the other, and why?

ANSWER:

Map:

- **Returns a new array:** map creates a new array by applying a provided function to each element of the original array.
- **Returns a result array:** The result of map is typically a new array with the same length as the original array.

ForEach:

- **forEach** iterates over the elements of the array and applies a provided function, but it doesn't return a new array.
- The **forEach** method returns **undefined**, and it's mainly used for its side effects.

Scenarios and Preferences:

1. **Use 'map' when you want a new array:**
    - If you need to create a new array based on the transformation of each element in an existing array, **map** is a better choice.

```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(number => number * number);
```

2. Use ‘forEach’ for side effects:

- If you only need to perform an action for each element without creating a new array, ‘forEach’ might be more appropriate.

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(number => console.log(number));
```

3. Performance Considerations:

- In terms of raw performance, ‘**forEach’** may be slightly faster than ‘**map’** because it doesn't have to create a new array. However, the difference might not be significant in most scenarios.

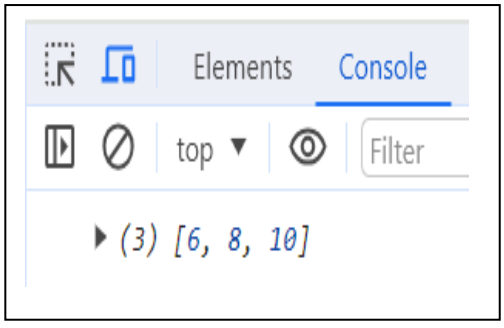
4. Consider readability and intent:

- Choose the method that best communicates your intent. If you are transforming data and creating a new array, use ‘**map’**. If you are iterating for side effects and not creating a new array, use ‘**forEach’**.

5. Chaining:

- **‘map’** is chainable, meaning you can easily chain other array methods like ‘**filter’** or ‘**reduce’** after it. This can make the code more concise and readable.

```
const numbers = [1, 2, 3, 4, 5];
const result = numbers
.map(number => number * 2)
.filter(number => number > 5);
console.log(result)
```



“Thank You”