ENSF 381
Full Stack Web Development

Lecture 13: Arrays and Objects

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Introduction to arrays.
- Common array methods.
- Looping through array elements.
- Objects.
- Spread operator.

What is Array?

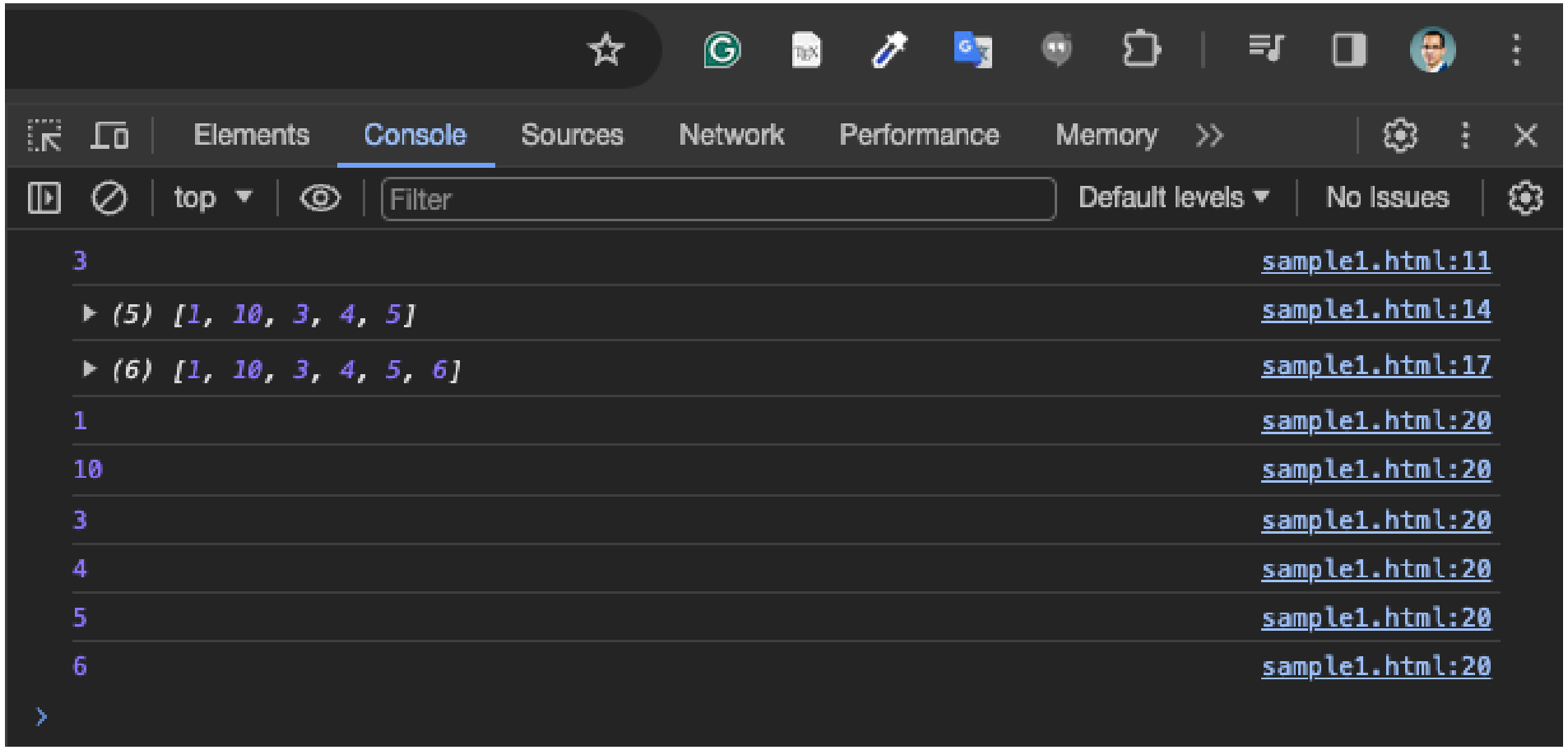
- A data structure that allows you to store and organize multiple values under a single name.

- These values, known as elements, can be of the same or different data types and are accessed using an index.

Arrays - example

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript String Example</title>
</head>
<body>
  <script>
// Creating an array of numbers
let numbers = [1, 2, 3, 4, 5];
// Accessing elements by index
console.log(numbers[2]); // Output: 3
// Modifying an element
numbers[1] = 10;
console.log(numbers); // Output: [1, 10, 3, 4, 5]
// Adding elements to the end of the array
numbers.push(6);
console.log(numbers); // Output: [1, 10, 3, 4, 5, 6]
// Iterating over the array
for (let i = 0; i < numbers.length; i++) {
console.log(numbers[i]);
}
</script></body></html>
```

Arrays - example



Arrays - methods

- `Length`: returns the number of elements in an array.

What is the output?

Code snippet

```
let fruits = ["apple", "orange",
"banana"];
console.log(fruits.length);
```

Output

→ 3

Arrays - methods

- `Length`: returns the number of elements in an array.

- `toString`: converts an array to a string by joining all elements with commas.

What is the output?

Code snippet

```
let fruits = ["apple", "orange",
"banana"];
console.log(fruits.length);
```

→ 3

```
let colors = ["red", "green", "blue"];
let colorsString = colors.toString();
console.log(colorsString);
```

→ red,green,blue

Output

Arrays - methods

- `Length`: returns the number of elements in an array.

- `toString`: converts an array to a string by joining all elements with commas.

- `Slice(start, end)`: returns **a shallow copy** of a portion of an array into a new array without modifying the original array.

What is the output?

Code snippet | **Output**

```
let fruits = ["apple", "orange",
"banana"];
console.log(fruits.length);
```

→ 3

```
let colors = ["red", "green", "blue"];
let colorsString = colors.toString();
console.log(colorsString);
```

→ red,green,blue

```
let numbers = [1, 2, 3, 4, 5];
let slicedNumbers = numbers.slice(1, 4);
console.log(slicedNumbers);
console.log(numbers);
```

→ [2, 3, 4]
[1, 2, 3, 4, 5]

Arrays - methods

- `Length`: returns the number of elements in an array.
- `toString`: converts an array to a string by joining all elements with commas.
- `Slice(start, end)`: returns **a shallow copy** of a portion of an array into a new array without modifying the original array.
- `Concat(arr)`: combines two or more arrays, creating a new array.

What is the output?

Code snippet

```
let arr1 = [1, 2, 3];
let arr2 = ["a", "b", "c"];
let combinedArray = arr1.concat(arr2);
console.log(combinedArray);
console.log(combinedArray.concat([4,5,6]));
```

Output

[1, 2, 3, "a", "b", "c"]
[1, 2, 3, "a", "b", "c", 4, 5, 6]

Arrays - methods

- `Length`: returns the number of elements in an array.
- `toString`: converts an array to a string by joining all elements with commas.
- `Slice(start, end)`: returns **a shallow copy** of a portion of an array into a new array without modifying the original array.
- `Concat(arr)`: combines two or more arrays, creating a new array.
- `Delete`: deletes an element at a specified index, but **leaves an empty slot**.

What is the output?

Code snippet

```
let arr1 = [1, 2, 3];
let arr2 = ["a", "b", "c"];
let combinedArray = arr1.concat(arr2);
console.log(combinedArray);
console.log(combinedArray.concat([4,5,6]));
```

Output

→ [1, 2, 3, "a", "b", "c"]
[1, 2, 3, "a", "b", "c", 4, 5, 6]

```
let numbers = [1, 2, 3, 4, 5];
delete numbers[2];
console.log(numbers);
```

→ [1, 2, empty, 4, 5]

Arrays - methods

- `Splice`: changes the contents of an array by removing or replacing existing elements and/or adding new elements.
  - The first parameter defines the position **where** new elements should be **added**.
  - The second parameter defines **how many** elements should be **removed**.

What is the output?

Code snippet

```
let fruits = ["apple", "orange", "banana"];
fruits.splice(1, 1, "grape", "kiwi");
console.log(fruits);
```

Output

→ ["apple", "grape", "kiwi", "banana"]

Arrays - methods

- `Splice`: changes the contents of an array by removing or replacing existing elements and/or adding new elements.
  - The first parameter defines the position **where** new elements should be **added**.
  - The second parameter defines **how many** elements should be **removed**.

- `Sort`: sorts the elements of an array in place (mutates the original array) based on Unicode values or a compare function.

- `Reverse`: reverses the order of the elements in an array in place (mutates the original array).

What is the output?

Code snippet | **Output**

```
let fruits = ["apple", "orange", "banana"];
fruits.splice(1, 1, "grape", "kiwi");
console.log(fruits);
```

→ ["apple", "grape", "kiwi", "banana"]

```
let fruits = ["banana", "apple", "orange"];
fruits.sort();
console.log(fruits);
```

→ ["apple", "banana", "orange"]

```
let numbers = [1, 2, 3, 4, 5];
numbers.reverse();
console.log(numbers);
```

→ [5, 4, 3, 2, 1]

Looping through array elements

For Loop

```
let fruits = ["apple", "banana", "orange"];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

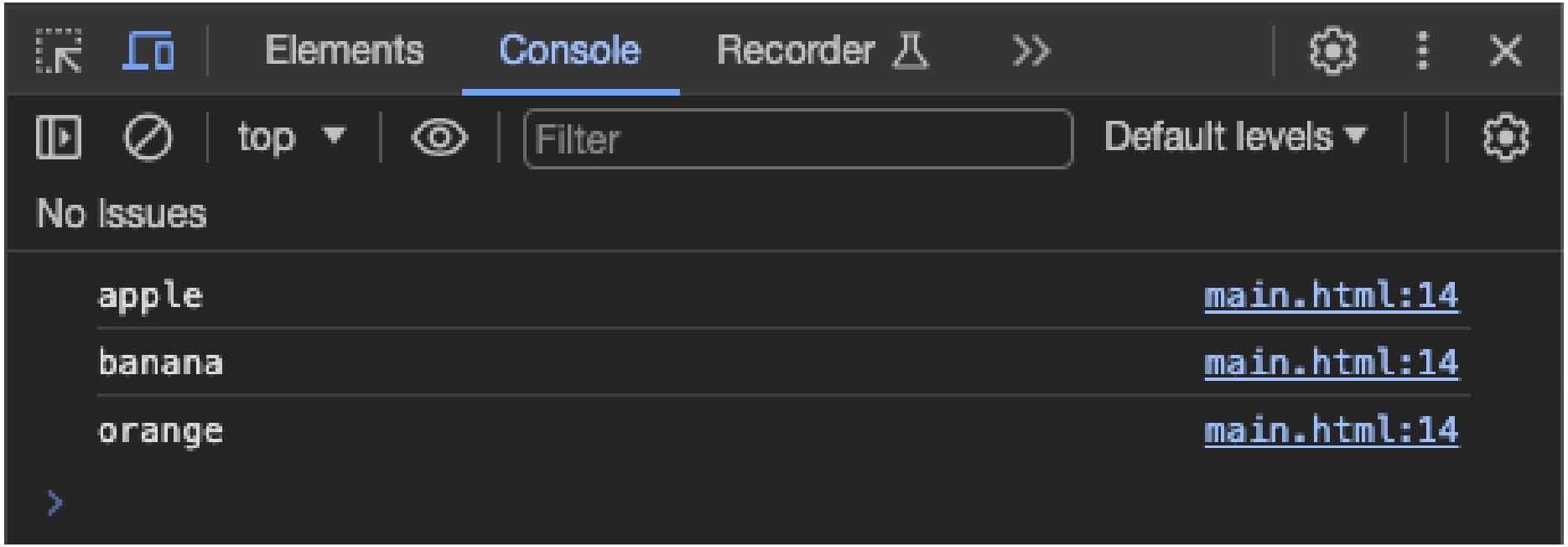
For...of Loop

```
let fruits = ["apple", "banana", "orange"];
for (let fruit of fruits) {
  console.log(fruit);
}
```

ForEach Method

```
let fruits = ["apple", "banana", "orange"];
fruits.forEach(function(fruit) {
  console.log(fruit);
});
```

Looping through array elements



Objects

- A complex data type that allows you to store and organize data in a structured way.
- Objects consist of key-value pairs.
- Objects are fundamental to the language and play a crucial role in representing and manipulating data.

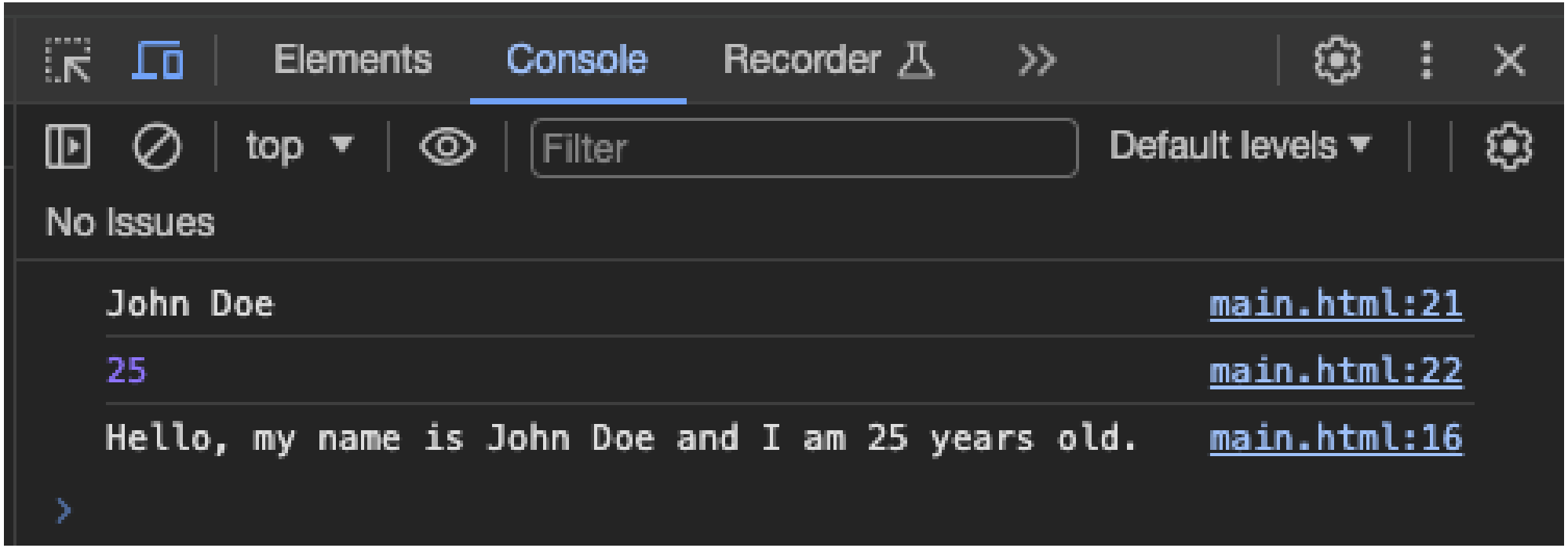
Objects - example

```html
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Object Example</title>
</head>
<body>
    <script>

    // Creating an object
    let person = {
      name: "John Doe",
      age: 25,
      isStudent: true,
      introduce: function() {
        console.log("Hello, my name is " + this.name + " and I am " + this.age + " years old.");
      }
    };

    // Accessing properties
    console.log(person.name);
    console.log(person["age"]);

    // Accessing methods
    person.introduce();
    </script>
</body></html>
```

You can also use:

```
person["name"];
person.age;
```

Objects - example



Recap: String template - example

- Allow for easy embedding of expressions within the string.
- Also known as template literals.
- The syntax for a template literal is ${expression}, where expression is any valid JavaScript expression.

```
let name = "John";
let age = 30;

let greeting = `Hello, my name is ${name} and I am ${age} years old.`;
console.log(greeting);
```

Object literal enhancement

A feature in modern JavaScript that provides a more concise way to define object literals.

Traditional Object Literal

```
let person = {
  name: "John",
  age: 25,
  greet: function() {
    console.log("Hello, my name is " + this.name
+ " and I am " + this.age + " years old.");
  }
};
```

Object Literal Enhancement

```
let person = {
  name: "John",
  age: 25,
  greet() {
    console.log(`Hello, my name is ${this.name} and
I am ${this.age} years old.`);
  }
};
```

Destructuring objects

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Object Example</title>
</head>
<body>
    <script>

    // Creating an object
    let person = {
      name: "John Doe",
      age: 25,
      isStudent: true,
      gender: "Male",
      major: "Software Engineering"
    };

    let name = person.name;
    let age = person.age;
    let isStudent = person.isStudent;
    let gender = person.gender;
    let major = person.major;

    </script>
</body></html>
```

Destructuring objects

- Allows to extract values from objects or arrays and assign them to variables in a more concise and convenient way.

- It provides a cleaner syntax for **extracting multiple properties** from an object and assigning them to variables.

- It is commonly used in scenarios where you need to extract specific properties from objects, especially when dealing with API responses or complex data structures.

Destructuring objects syntax

```
let { var1, var2 } = object;
```

- var1 and var2 are the names of the properties you want to extract from the object
- The variable names on the left side of the assignment **should match** the property names in the object.

Destructuring objects - example

```
// Creating an object
let person = {
  name: "John Doe",
  age: 25,
  address: {
    city: "Example City",
    country: "Example Country"
  }
};

// Destructuring object properties
let { name, age, address } = person;

console.log(name); // Output: John Doe
console.log(age); // Output: 25
console.log(address); // Output: { city: 'Example City', country: 'Example Country'}
```

Destructuring objects – renaming variables example

```
// Creating an object
let person = {
  name: "John Doe",
  age: 25,
  address: {
    city: "Example City",
    country: "Example Country"
  }
};

// Destructuring object properties
let { name: personName, age: personAge} = person;

console.log(personName); // Output: John Doe
console.log(personAge); // Output: 25
```

Destructuring arrays - example

Destructuring assignment in JavaScript is not limited to objects; it can also be applied to arrays:

```
let [element1, element2] = array;
```

Example:

```
const [firstAnimal] = ["Horse", "Mouse", "Cat"];
console.log(firstAnimal); // Horse
```

We can pass over unnecessary elements with list:

```
const [, , thirdAnimal] = ["Horse", "Mouse", "Cat"];
console.log(thirdAnimal); // Cat
```

Spread operator

- A syntax in JavaScript "**...**" that allows an iterable (like an array or a string) to be expanded or spread into individual elements.

- It has several use cases:
  - Copying arrays and objects: easily create copies of arrays and objects **without modifying the original**.
  - Combine multiple arrays into a single array.

Spread operator - example

Copying arrays example:

```
let originalArray = [1, 2, 3];
let copiedArray = [...originalArray];

console.log(copiedArray); // Output: [1, 2, 3]
```

Combining arrays example:

```
let array1 = [1, 2, 3];
let array2 = [4, 5, 6];
let combinedArray = [...array1, ...array2];

console.log(combinedArray); // Output: [1, 2, 3, 4, 5, 6]
```

Question…

What is the output of the following code snippet:

```
let originalArray = [1, 2, 3];
let copiedArray = [...originalArray];
copiedArray[0] = 10
console.log(copiedArray);
console.log(originalArray);
```

Output:

[10, 2, 3]

[1, 2, 3]

Questions

Reminder: The deadline for Assignment 1 is **today at 11:59 PM**.