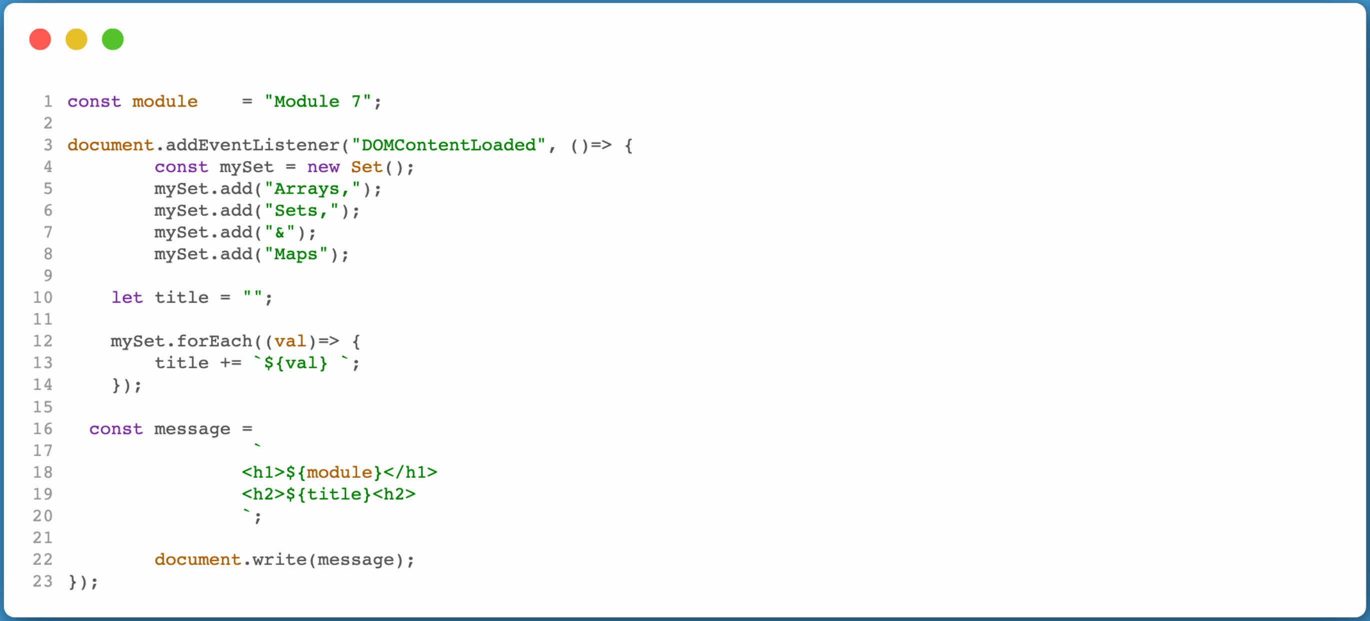
INFO 2124

JavaScript



```
const module    = "Module 7";

document.addEventListener("DOMContentLoaded", ()=> {
        const mySet = new Set();
        mySet.add("Arrays,");
        mySet.add("Sets,");
        mySet.add("&");
        mySet.add("Maps");

    let title = "";

    mySet.forEach((val)=> {
        title += `${val} `;
    });

  const message =
                 `
                <h1>${module}</h1>
                <h2>${title}<h2>
                `;

        document.write(message);
});
```

Speaker notes

speaker notes

Module Objectives

A. Describe the creation and use of an array.

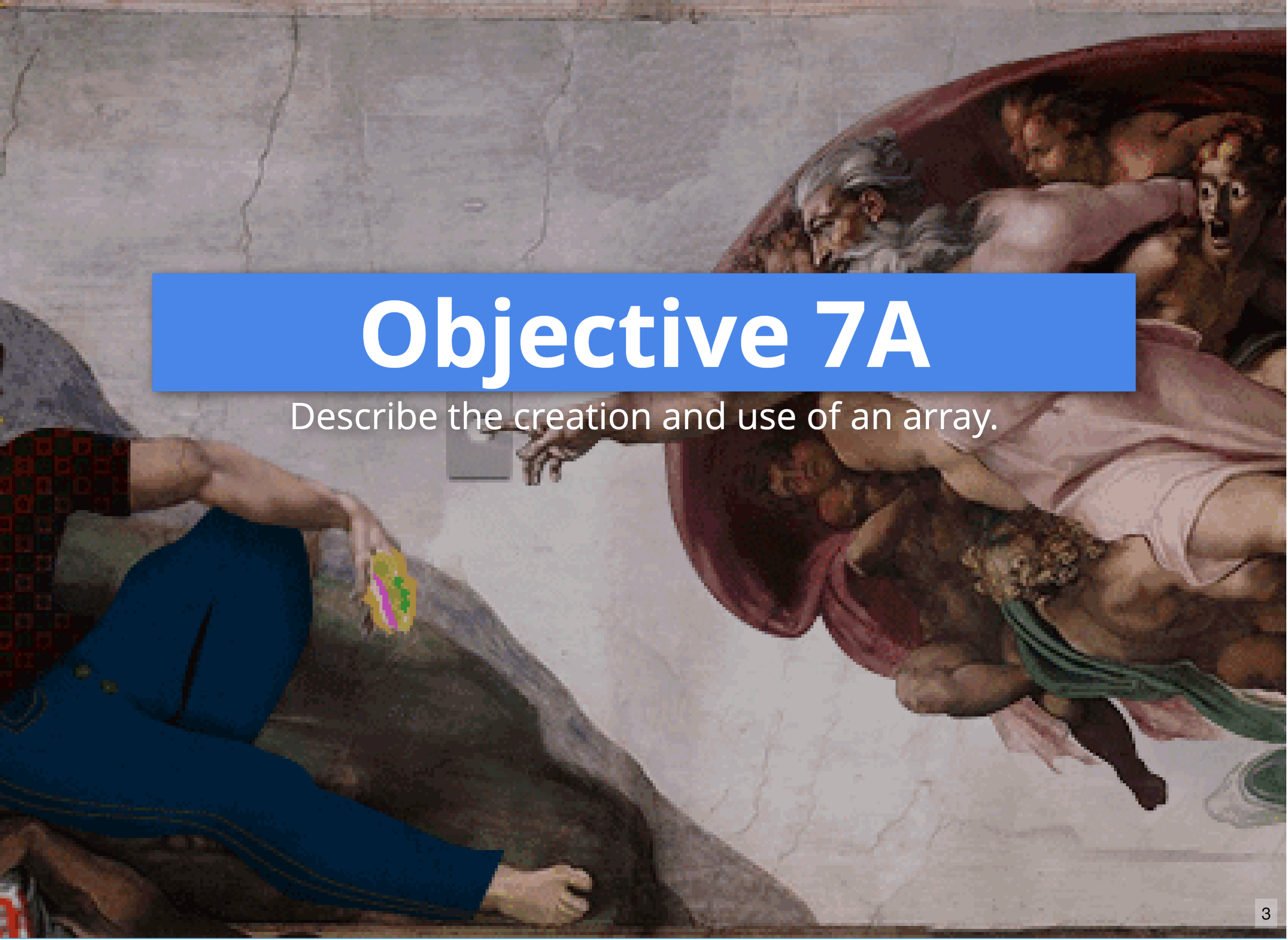
B. Describe these methods of an Array object: push(), pop(), unshift(), splice(), slice(), indexOf(), lastIndexOf(), includes(), entries(), values(), keys(), map(), filter(), and reduce().

C. Distinguish between an array, a set, and a map.

D. Use arrays in your applications.

E. Use associative arrays and arrays of arrays in your applications.

F. Use sets and maps in your applications.



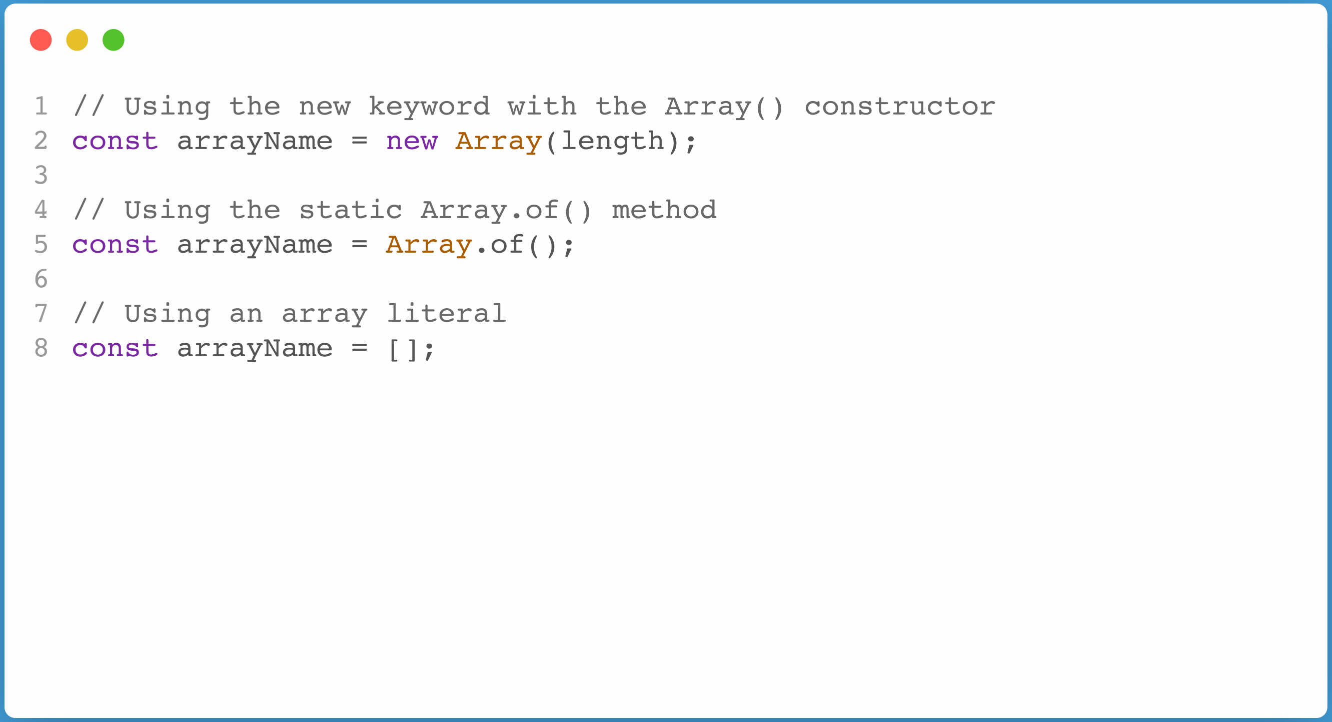
Objective 7A

Describe the creation and use of an array.

Arrays . . .

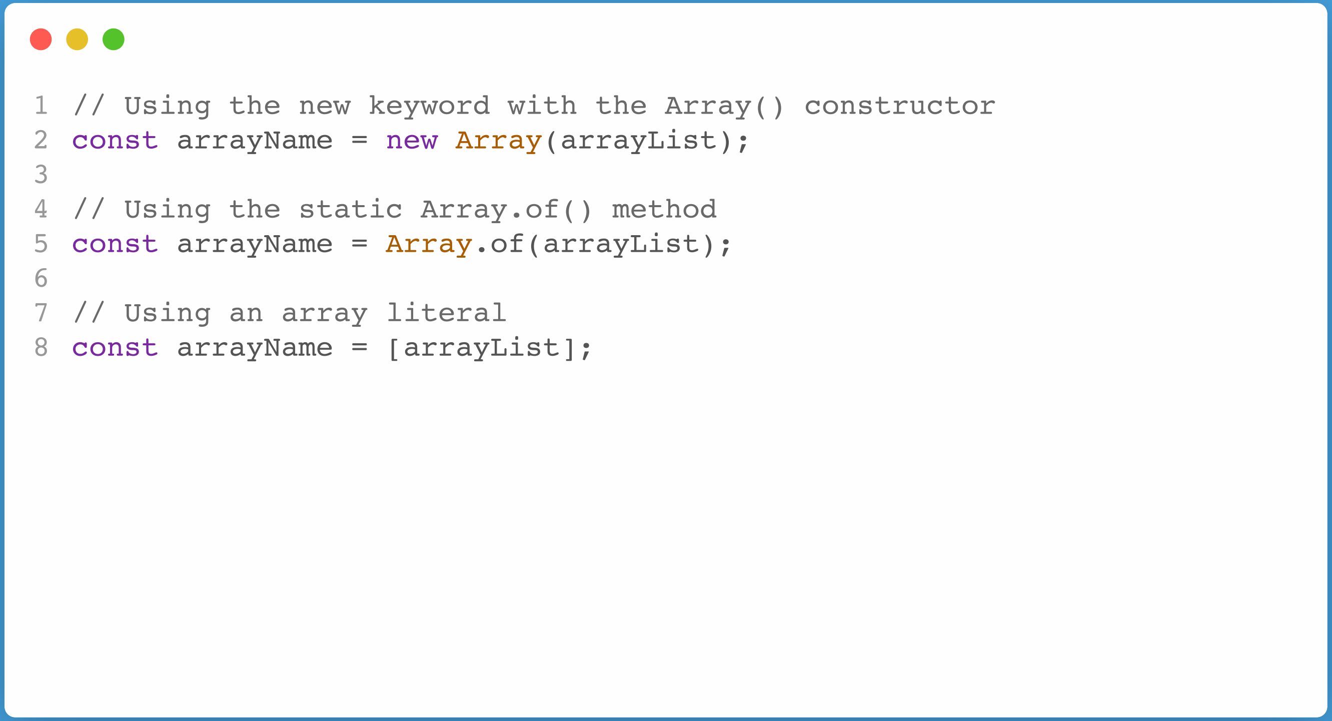
- An *array* can store on or more *elements*.
- An array's *length* is the number of elements in the array.
- If you create an array without specifying the length, the array doesn't contain any elements (it's empty).
- When you create an array of one or more elements without assigning values to them, each element is set to undefined.
- To refer to the elements in an array, you use an *index* where 0 is the first element, 1 is the second element, and so on . . .

The Syntax for Creating an Array



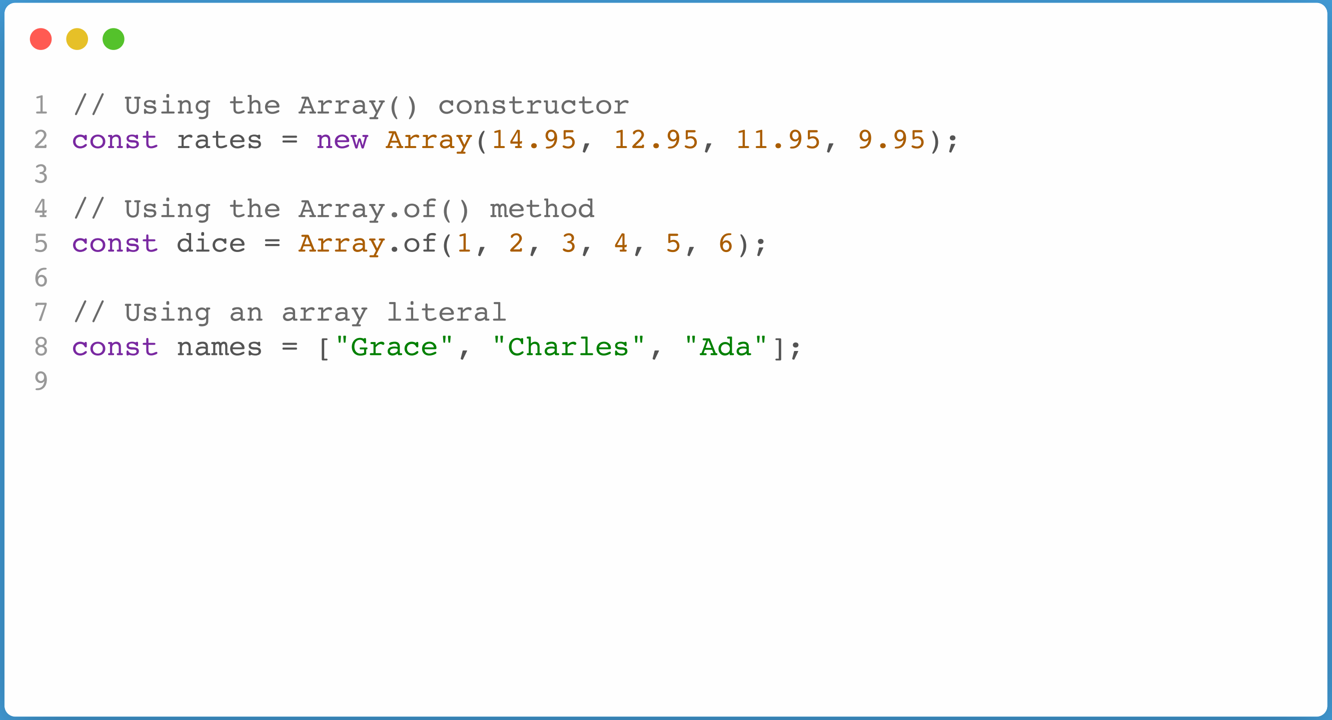
```javascript
// Using the new keyword with the Array() constructor
const arrayName = new Array(length);

// Using the static Array.of() method
const arrayName = Array.of();

// Using an array literal
const arrayName = [];
```

The Syntax For Creating An Array And Assigning Values In One Statement



```javascript
// Using the new keyword with the Array() constructor
const arrayName = new Array(arrayList);

// Using the static Array.of() method
const arrayName = Array.of(arrayList);

// Using an array literal
const arrayName = [arrayList];
```

Examples That Create An Array And Assign Values In One Statement

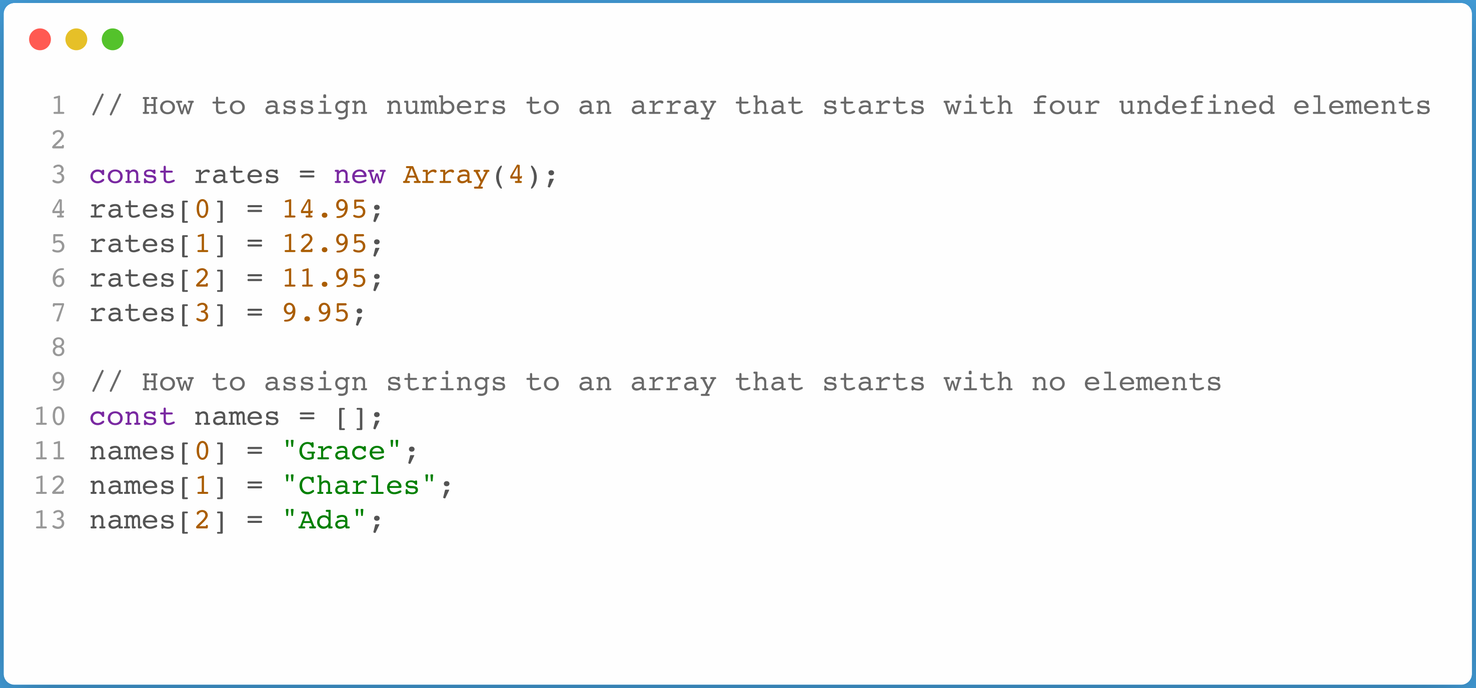


```javascript
// Using the Array() constructor
const rates = new Array(14.95, 12.95, 11.95, 9.95);

// Using the Array.of() method
const dice = Array.of(1, 2, 3, 4, 5, 6);

// Using an array literal
const names = ["Grace", "Charles", "Ada"];

```

Working With Arrays

```
// The syntax for referring to an element of an array

arrayName[index]

// Code that refers to the elements in an array
rates[2]    // Refers to third element in rates array
names[1]    // Refers to second element in names array
```

How To Assign Values To An Array By Accessing Each Element



```javascript
// How to assign numbers to an array that starts with four undefined elements

const rates = new Array(4);
rates[0] = 14.95;
rates[1] = 12.95;
rates[2] = 11.95;
rates[3] = 9.95;

// How to assign strings to an array that starts with no elements
const names = [];
names[0] = "Grace";
names[1] = "Charles";
names[2] = "Ada";
```

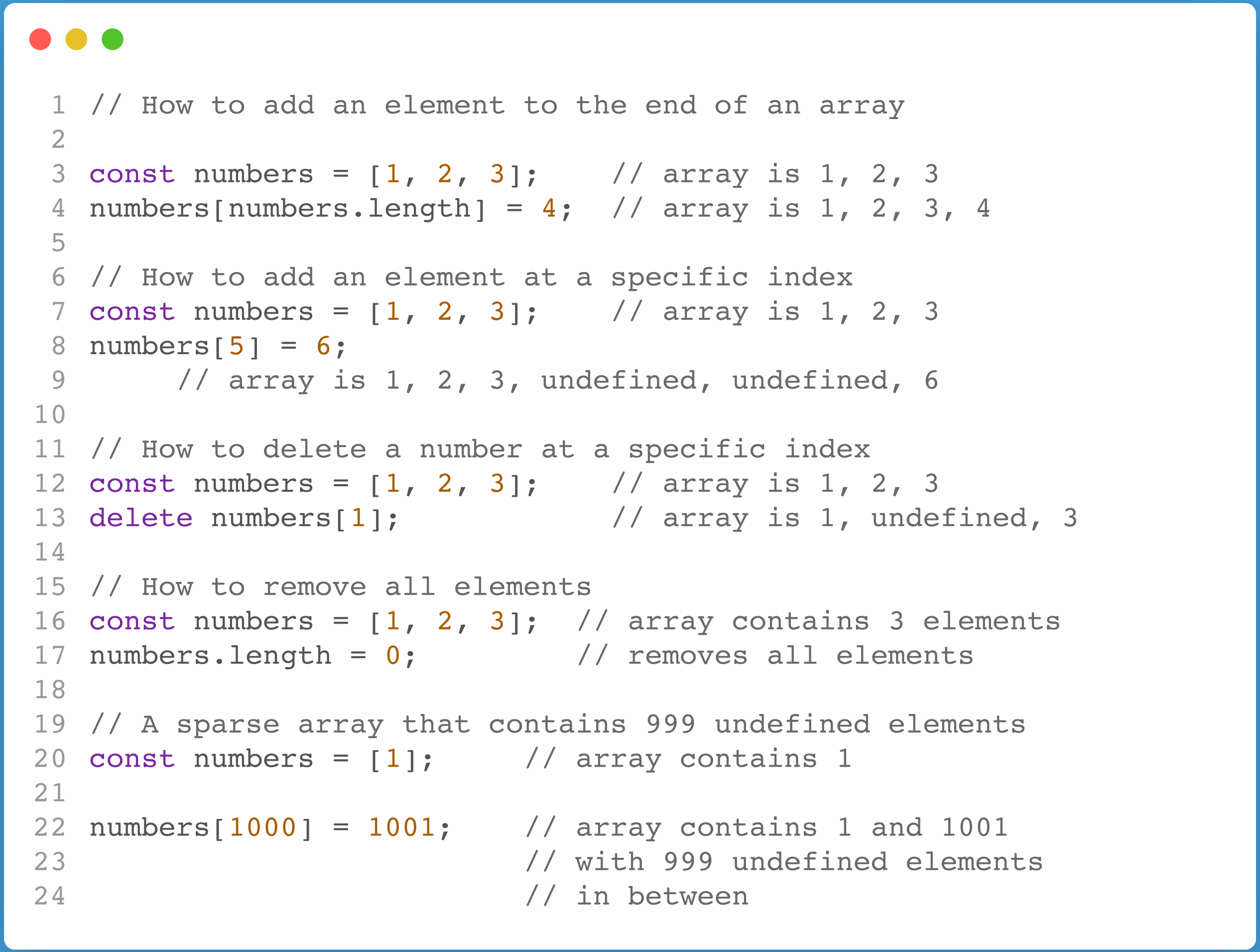
Working With Arrays

- You can add an element to the end of an array by using the *length* property as the index.
- If you add an element at a specific index that isn't the next one in the sequence, JavaScript adds undefined elements to the array between the new element and the end of the original array.
- When you use the delete operator to delete an element, JavaScript deletes the element's value but keeps the element in the array with a value of undefined.
- To remove all the elements from an array, you can set the array's length property to zero. Unlike the delete operator, this removes all the elements, not just the element's values.
- A *sparse array* is a large array with few defined elements. For efficiency, though, JavaScript only reserves space for the elements that are assigned values.

Working With Arrays

- One property of an array and the delete operator:
  - **length**
    - A property of an array, which specifies the number of elements in the array.
  - **delete**
    - An operator, which deletes the contents of an element and sets the element to undefined, but *doesn't* remove the element from the array.

Working With Arrays

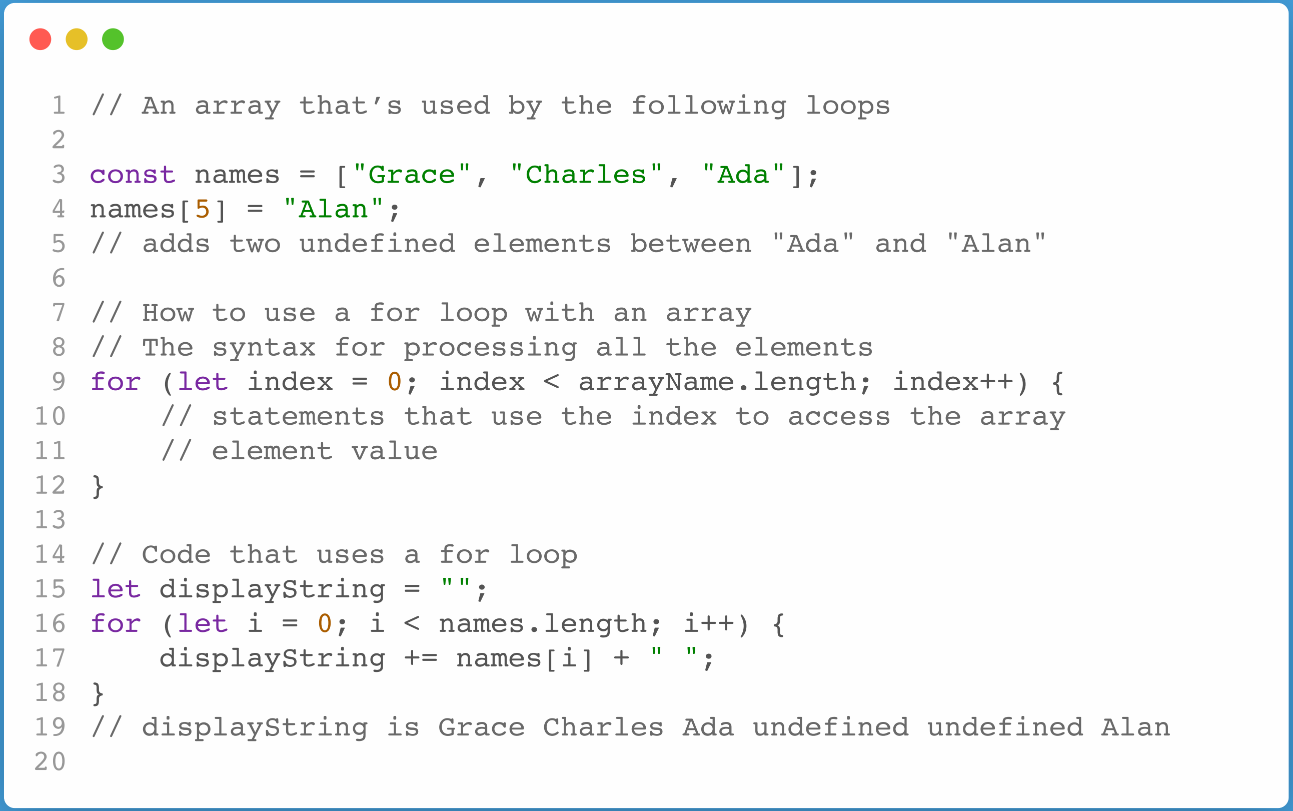


```javascript
// How to add an element to the end of an array

const numbers = [1, 2, 3];     // array is 1, 2, 3
numbers[numbers.length] = 4;  // array is 1, 2, 3, 4

// How to add an element at a specific index
const numbers = [1, 2, 3];     // array is 1, 2, 3
numbers[5] = 6;
     // array is 1, 2, 3, undefined, undefined, 6

// How to delete a number at a specific index
const numbers = [1, 2, 3];     // array is 1, 2, 3
delete numbers[1];             // array is 1, undefined, 3

// How to remove all elements
const numbers = [1, 2, 3];   // array contains 3 elements
numbers.length = 0;          // removes all elements

// A sparse array that contains 999 undefined elements
const numbers = [1];     // array contains 1

numbers[1000] = 1001;    // array contains 1 and 1001
                         // with 999 undefined elements
                         // in between
```

Looping Through Arrays

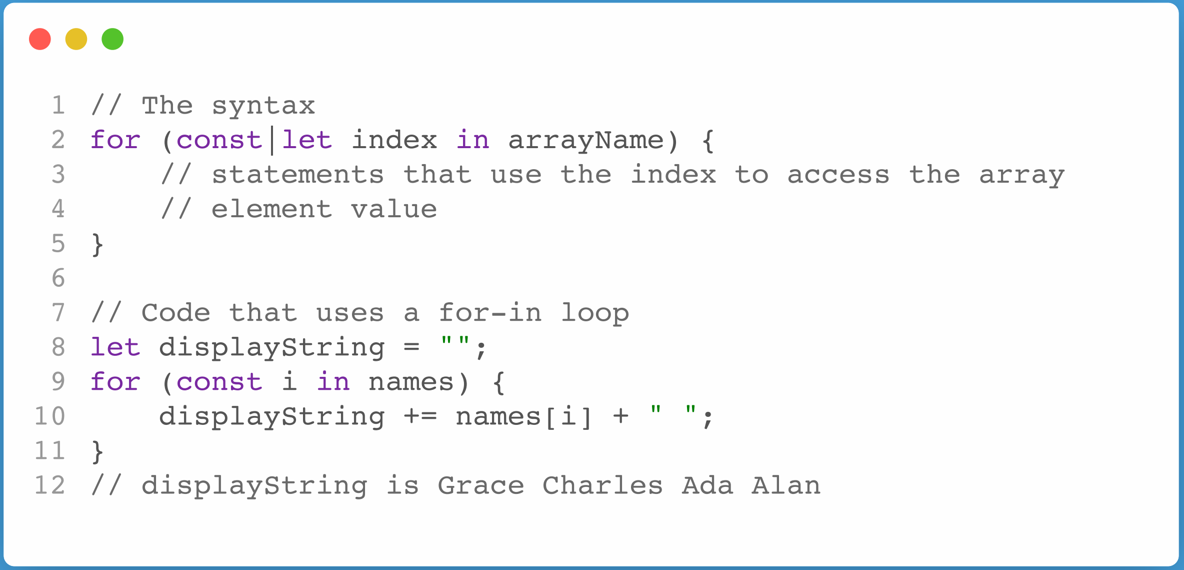
- You can use a *for loop*, a *for-in loop*, or a *for-of loop* to iterate through the elements of an array.
- You can use a constant or a variable with a for-in or for-of statement to refer to an element index or value. By contrast, the counter in a for statement must be a variable.
- The constant or variable that's declare in a for-in statement is the index of the current element, and the constant or variable that's declared in a for-of statement is the value of the current element.
- A for-in loop doesn't process undefined elements, but for and for-of loops do.

Working With Arrays



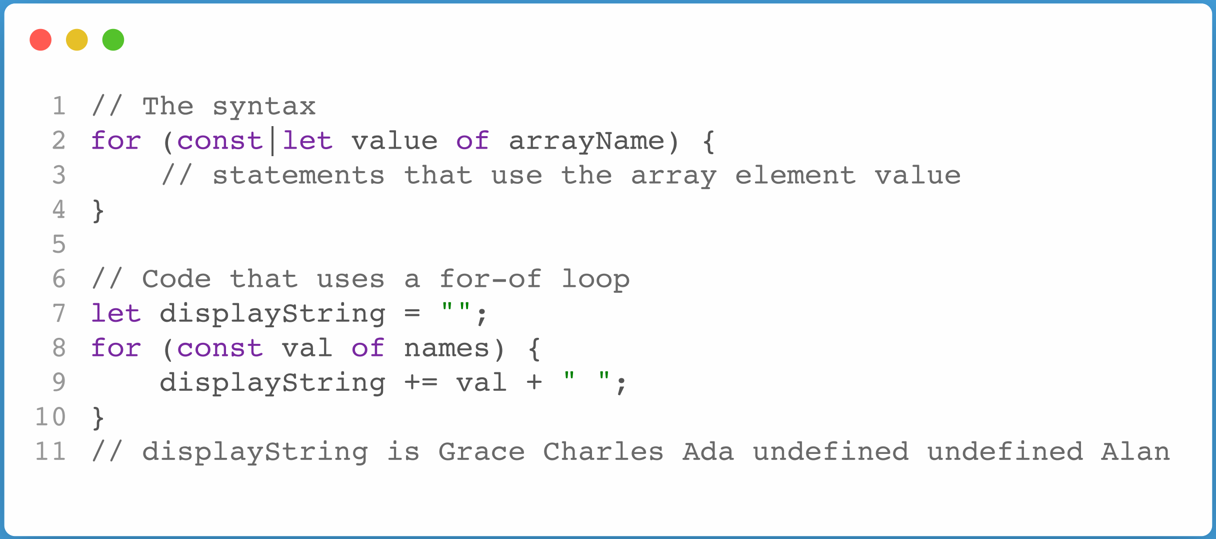
```
// An array that's used by the following loops

const names = ["Grace", "Charles", "Ada"];
names[5] = "Alan";
// adds two undefined elements between "Ada" and "Alan"

// How to use a for loop with an array
// The syntax for processing all the elements
for (let index = 0; index < arrayName.length; index++) {
    // statements that use the index to access the array
    // element value
}

// Code that uses a for loop
let displayString = "";
for (let i = 0; i < names.length; i++) {
    displayString += names[i] + " ";
}
// displayString is Grace Charles Ada undefined undefined Alan

```

How To Use A For-in Loop With An Array



```
// The syntax
for (const|let index in arrayName) {
    // statements that use the index to access the array
    // element value
}

// Code that uses a for-in loop
let displayString = "";
for (const i in names) {
    displayString += names[i] + " ";
}
// displayString is Grace Charles Ada Alan
```

How To Use A For-in Loop With An Array



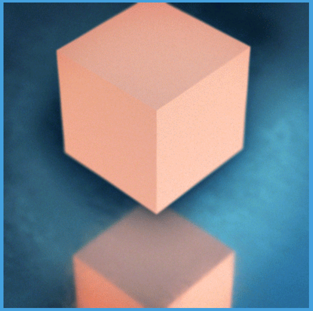
```
// The syntax
for (const|let value of arrayName) {
    // statements that use the array element value
}

// Code that uses a for-of loop
let displayString = "";
for (const val of names) {
    displayString += val + " ";
}
// displayString is Grace Charles Ada undefined undefined Alan
```

Working With Arrays

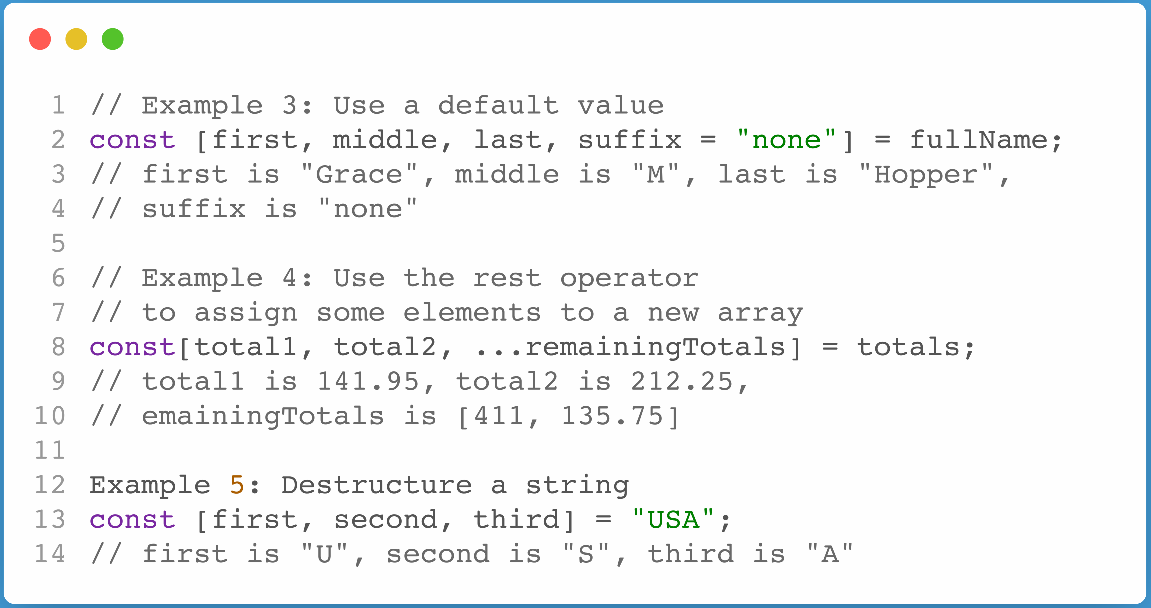
```
// The syntax for destructuring an array
const|let [identifier1, identifier2, ...] = arrayName;

// Two arrays used by the following examples
const totals = [141.95, 212.25, 411, 135.75];
const fullName = ["Grace", "M", "Hopper"];

// Example 1: Assign the first three elements
// in an array
const [total1, total2, total3] = totals;
// total1 is 141.95, total2 is 212.25, total3 is 411

// Example 2: Skip an array element
const [firstName, , lastName] = fullName;
// firstName is "Grace", lastName is "Hopper"
```

Destructuring Arrays

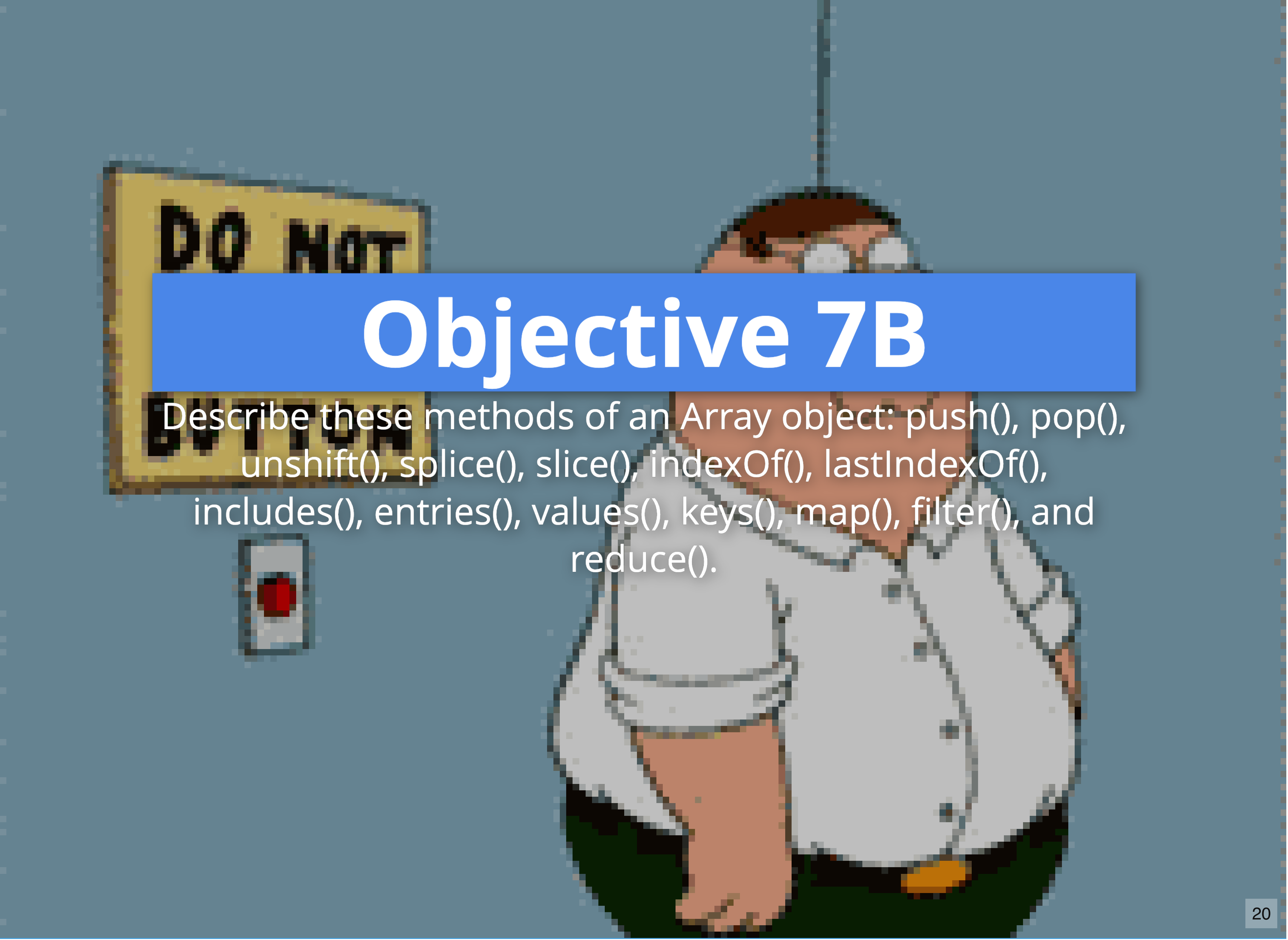
- You can assign the elements in an array to individual constants or variables by coding the *const* or *let* keywords, followed by a list of variable or constant names enclosed in brackets ([ ]), an equals sign, and the name of the array. This is called *destructuring* an array.
- When you destructure an array, you can skip elements by coding commas as placeholders. You can also specify default values that are assigned if an element doesn't exist.
- When you destructure an array, you can use the *rest operator* (...) to assign the rest of the elements in an array to another array. *Destructuring* and the rest operator were introduced with ECMAScript 2015 and can't be used with older browsers.
- You can also destructure the individual characters of a string.
- When you destructure an array or a string, the original array or string isn't changed.



Working With Arrays



```
// Example 3: Use a default value
const [first, middle, last, suffix = "none"] = fullName;
// first is "Grace", middle is "M", last is "Hopper",
// suffix is "none"

// Example 4: Use the rest operator
// to assign some elements to a new array
const[total1, total2, ...remainingTotals] = totals;
// total1 is 141.95, total2 is 212.25,
// emainingTotals is [411, 135.75]

Example 5: Destructure a string
const [first, second, third] = "USA";
// first is "U", second is "S", third is "A"
```



Objective 7B

Describe these methods of an Array object: push(), pop(), unshift(), splice(), slice(), indexOf(), lastIndexOf(), includes(), entries(), values(), keys(), map(), filter(), and reduce().

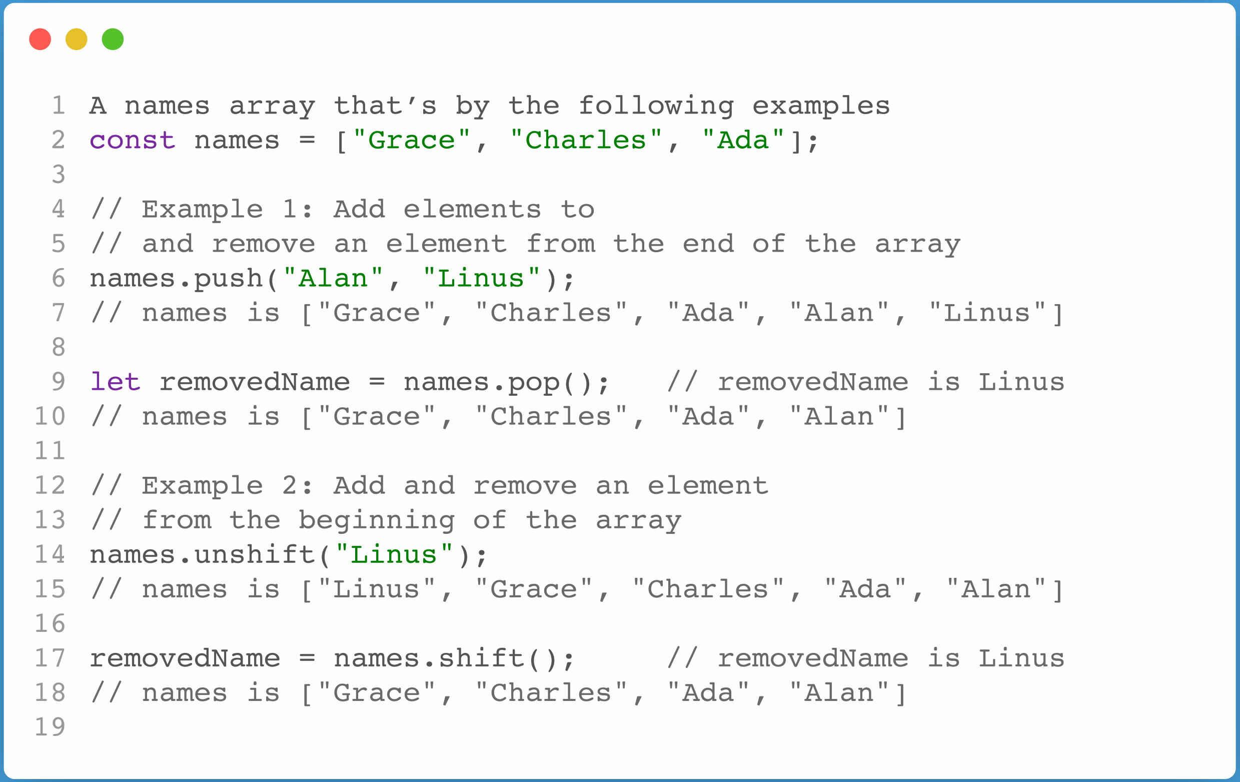
Methods Of The Array Type That Add, Modify, Remove, And Copy Elements

- **push(***element_list***)**
  - Adds one or more elements to the end of the array and returns the new length of the array.
- **pop()**
  - Removes the *last* element in the array, decrements the length, and returns the element that is removed.
- **unshift(***element_list***)**
  - Adds one or more elements to the beginning of the array, shifts other elements to the right, and returns the new length.
- **splice(***start, number***)**
  - Removes the number of elements given by the second parameter starting with the index given by the first parameter. It returns the elements that were removed.

Methods Of The Array Type That Add, Modify, Remove, And Copy Elements

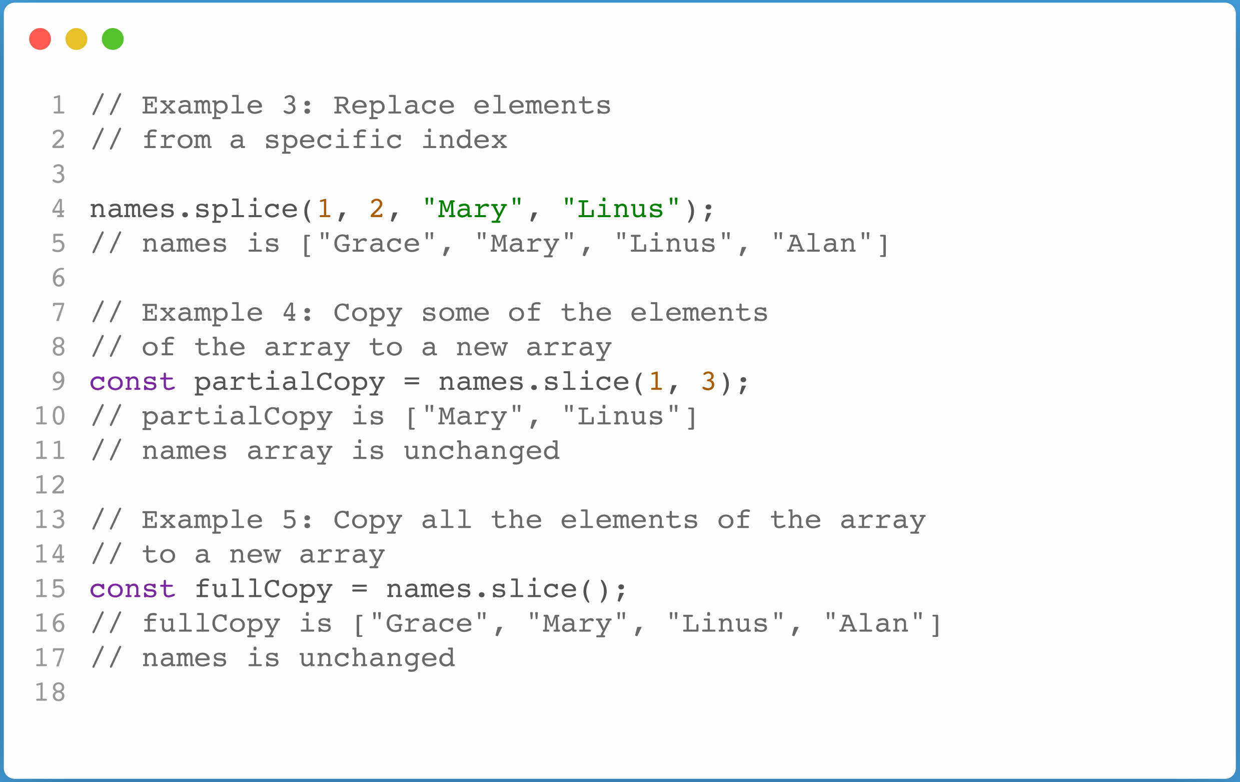
- **splice(***start, number, element_list***)**
  - Removes the number of elements given by the second parameter starting with the index given by the first parameter, and replaces those elements with the ones given by the third parameter.
    - If the second parameter is 0, the elements are added at the start index instead of being replaced. It returns the elements that were removed.
- **slice(***[start] [, end]***)**
  - Returns a new array that start with the index given by the first parameter and ends with the element before the index given in the second parameter. It doesn't change the original array.

Working With Arrays



```
A names array that’s by the following examples
const names = ["Grace", "Charles", "Ada"];

// Example 1: Add elements to
// and remove an element from the end of the array
names.push("Alan", "Linus");
// names is ["Grace", "Charles", "Ada", "Alan", "Linus"]

let removedName = names.pop();    // removedName is Linus
// names is ["Grace", "Charles", "Ada", "Alan"]

// Example 2: Add and remove an element
// from the beginning of the array
names.unshift("Linus");
// names is ["Linus", "Grace", "Charles", "Ada", "Alan"]

removedName = names.shift();      // removedName is Linus
// names is ["Grace", "Charles", "Ada", "Alan"]

```

Working With Arrays



```
// Example 3: Replace elements
// from a specific index

names.splice(1, 2, "Mary", "Linus");
// names is ["Grace", "Mary", "Linus", "Alan"]

// Example 4: Copy some of the elements
// of the array to a new array
const partialCopy = names.slice(1, 3);
// partialCopy is ["Mary", "Linus"]
// names array is unchanged

// Example 5: Copy all the elements of the array
// to a new array
const fullCopy = names.slice();
// fullCopy is ["Grace", "Mary", "Linus", "Alan"]
// names is unchanged

```

Methods Of The Array Type That Inspect An Array Or Its Elements

- **indexOf(***value[, start]***)**
  - Returns the first index at which the first parameter is found or -1 if the value isn't found. The optional second parameter specifies the index to start searching from.
- **lastIndexOf(***value[, start]***)**
  - Returns the last index at which the first parameter is found, or -1 if the value isn't found. The optional second parameter specifies the index to start searching from.
- **includes(***value[, start]***)**
  - Returns a Boolean value that indicates whether the specified value is in the array. The optional second parameter specifies the index to start searching from.

Methods Of The Array Type That Inspect An Array Or Its Elements

- **find(*function*)**
  - Returns the value of the first element that meets the condition of the function, or undefined if no element meets the condition.
- **findIndex(*function*)**
  - Returns the index of the first element that meets the condition of the function, or -1 if no element meets the condition.
- **filter(*function*)**
  - Returns an array containing all the elements that meet the condition of the function.
- **every(*function*)**
  - Returns a Boolean value that indicates whether all elements meet the condition of the function.

Methods Of The Array Type That Inspect An Array Or Its Elements

- **some(*function*)**
  - Returns a Boolean value that indicates whether at least one element meets the condition of the function.
- **entries()**
  - Returns an iterator that contains a [key, value] array for each element. The key is the index of the element.
- **values()**
  - Returns an iterator that contains the value of each element.

"Iterator"

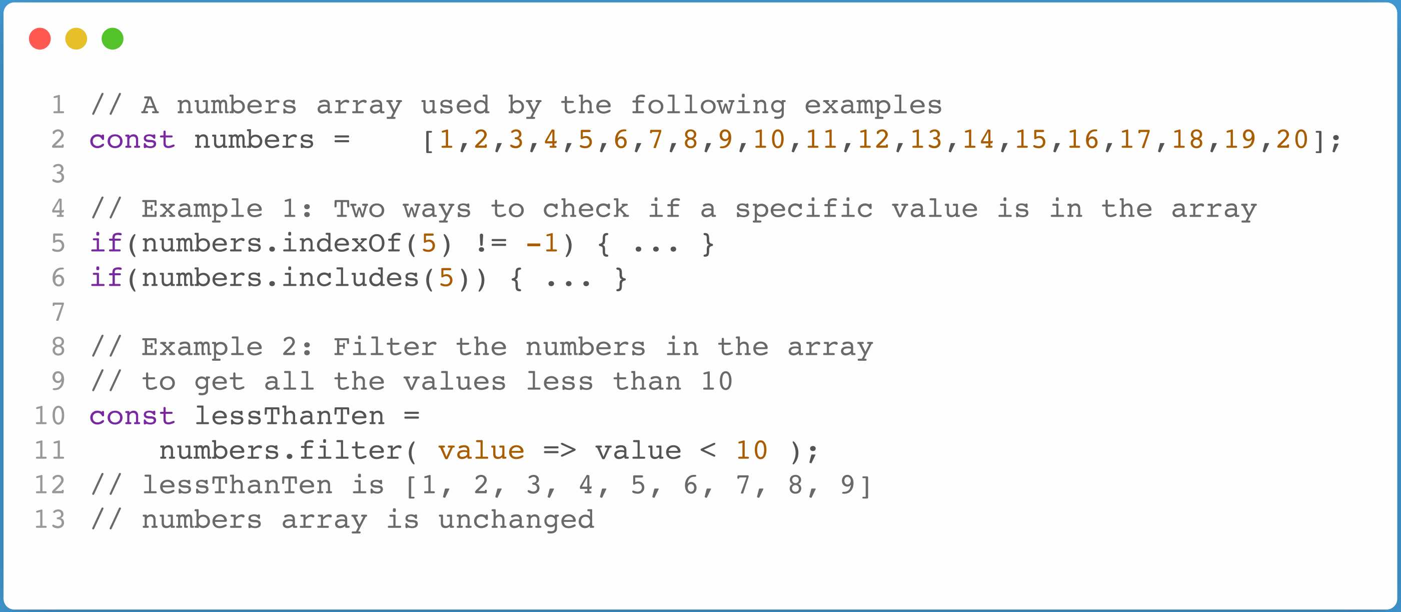
- An iterator is an object that can be used to *iterate* (or "loop") through the items in a collection.
- MDN defines the term as follows:
  - "In JavaScript an iterator is an object which defines a sequence and potentially a return value upon its termination.

    Specifically, an iterator is any object which implements the Iterator protocol by having a next() method that returns an object with two properties: [value and done]."
- Basically, an iterator is a function that "knows" how to loop through a specific type of collection.
- Iterators will be discussed in a subsequent module but it's important to at least define the term, since it's being used in this presentation.

Methods Of The Array Type That Inspect An Array Or Its Elements

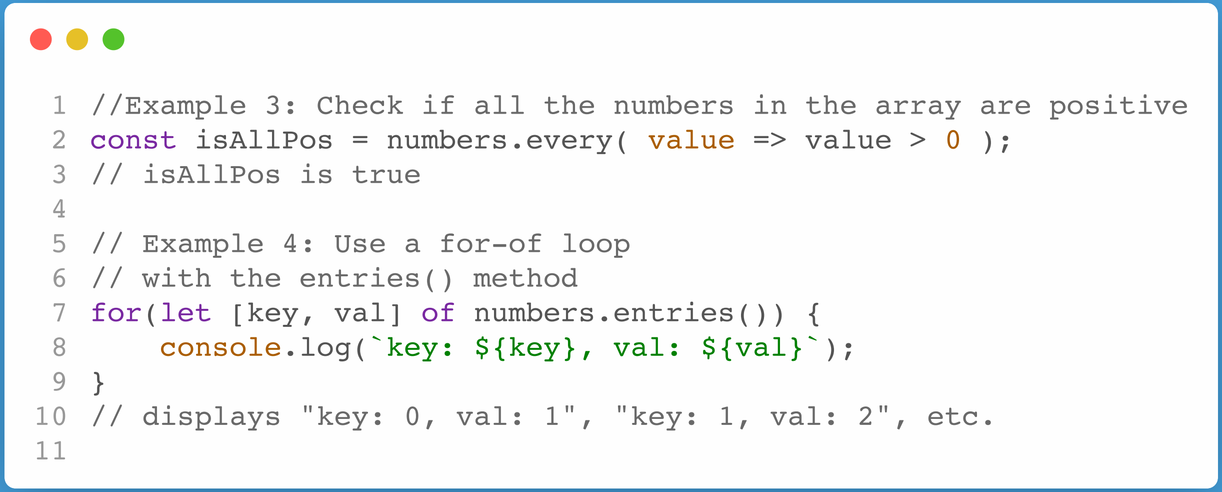
- **keys()**
  - Returns an iterator that contains a key for each element.
- **isArray(***object***)**
  - A static method that returns a Boolean value that indicates whether the specified object is an array.

Methods Of The Array Type That Inspect An Array Or Its Elements



```javascript
// A numbers array used by the following examples
const numbers =    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];

// Example 1: Two ways to check if a specific value is in the array
if(numbers.indexOf(5) != -1) { ... }
if(numbers.includes(5)) { ... }

// Example 2: Filter the numbers in the array
// to get all the values less than 10
const lessThanTen =
    numbers.filter( value => value < 10 );
// lessThanTen is [1, 2, 3, 4, 5, 6, 7, 8, 9]
// numbers array is unchanged
```

Methods Of The Array Type That Inspect An Array Or Its Elements



```
//Example 3: Check if all the numbers in the array are positive
const isAllPos = numbers.every( value => value > 0 );
// isAllPos is true

// Example 4: Use a for-of loop
// with the entries() method
for(let [key, val] of numbers.entries()) {
    console.log(`key: ${key}, val: ${val}`);
}
// displays "key: 0, val: 1", "key: 1, val: 2", etc.

```

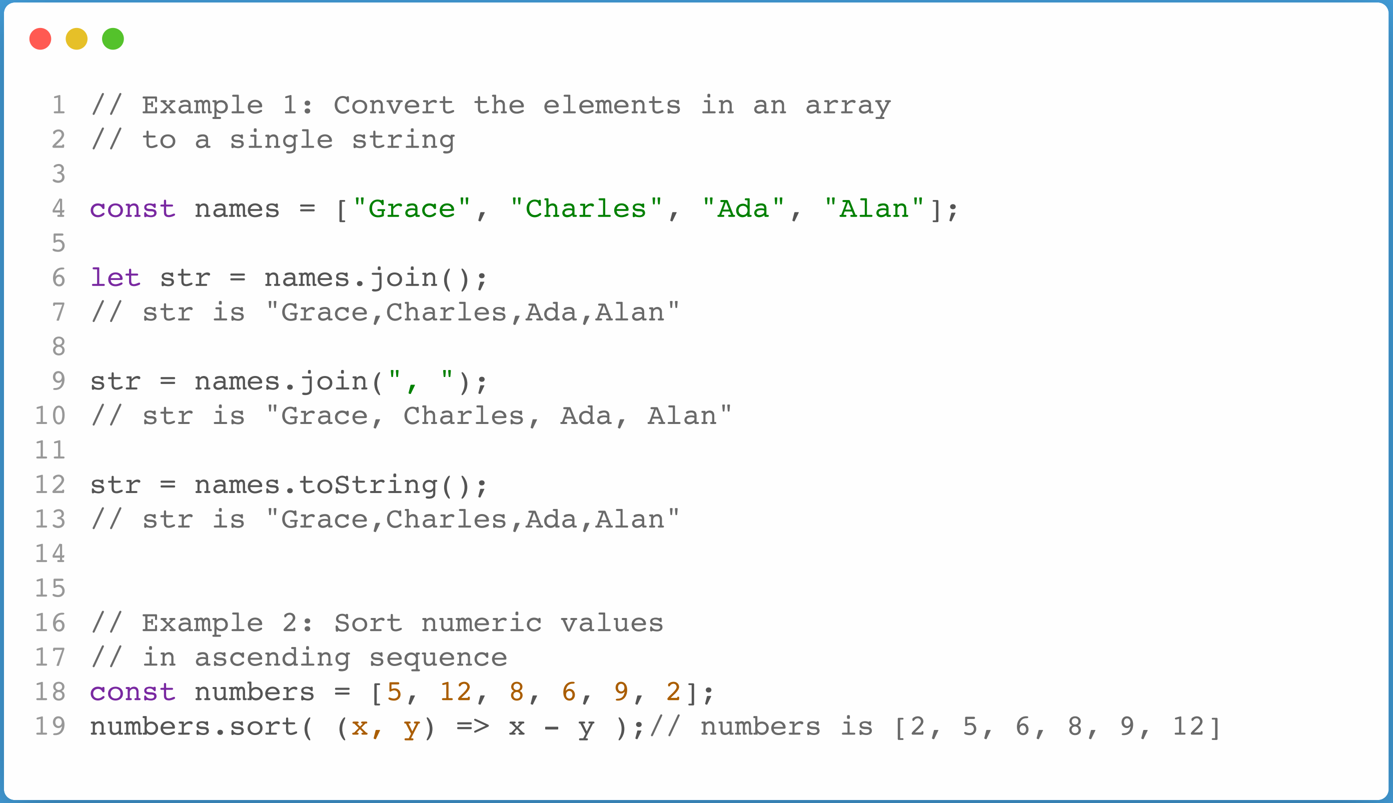
Methods Of The Array Type That Transform Elements

- **forEach(***function***)**
  - Executes the function one for each element and returns a value of undefined.
- **join(***[separator]***)**
  - Converts all the elements of the array to strings and concatenates them separated by commas or by the string value of the parameter if one is specified.
- **toString()**
  - Same as the join() method without any parameter passed to it.
- **sort(***[comparison_function]*)
  - Sorts the elements into ascending alphanumeric sequence, converting numeric elements to strings if necessary. Accepts an optional function to change the default sort order.
- **reverse()**
  - Reserves the order of the elements.

Methods Of The Array Type That Transform Elements

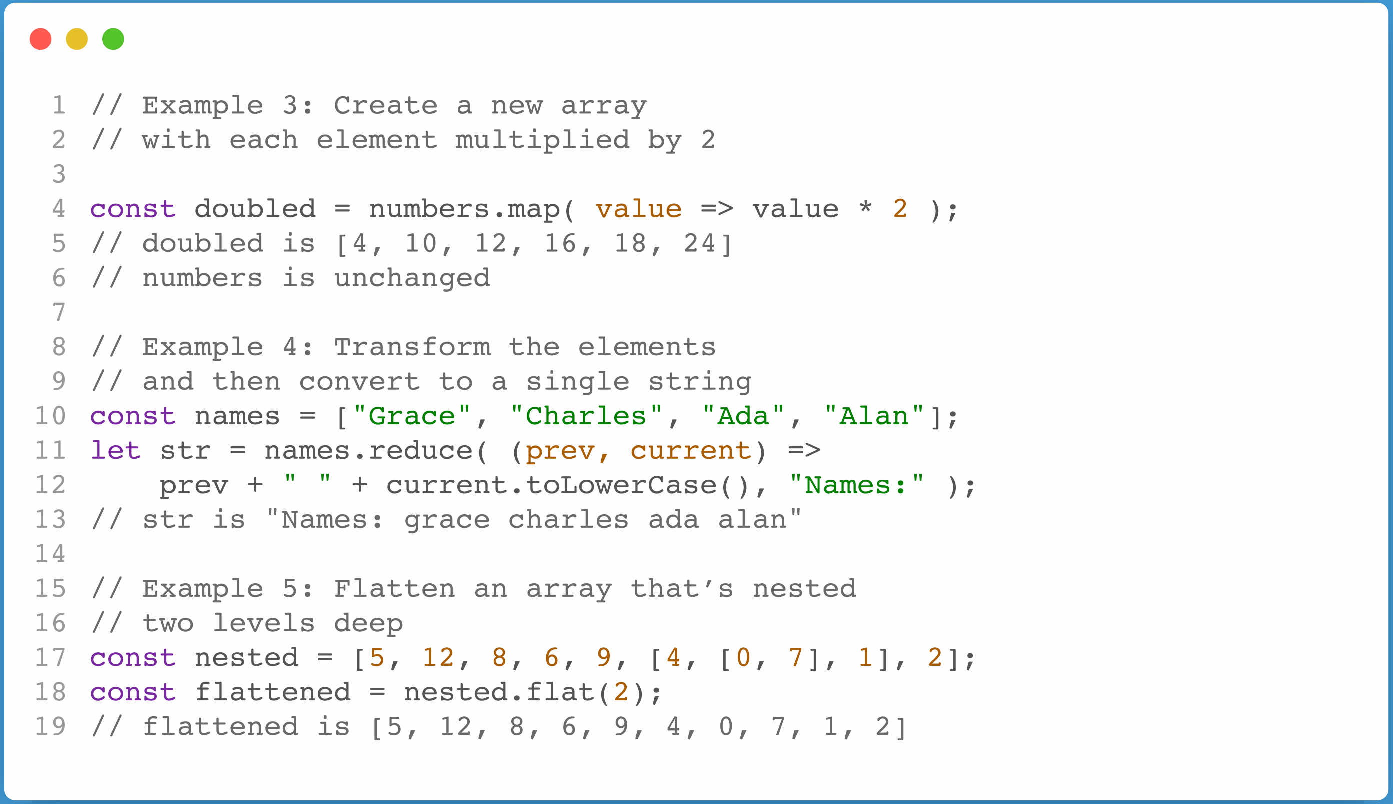
- **map(***function***)**
  - Executes the function once for each element and returns an array that contains the results of each function call.
- **reduce(***function[, init]***)**
  - Executes a function that returns all the elements reduced to one value, processed in ascending sequence. The optional second parameter sets the initial value for the function.
- **flat(***[depth]***)**
  - Returns an array with any nested array elements included. The optional parameter is the number of levels to flatten. The default is 1.
- **flatMap(***function***)**
  - Returns an array with the results of the function call, flattened to one level. Equivalent to calling *flat()* after calling *map()*.

Methods Of The Array Type That Transform Elements



```
// Example 1: Convert the elements in an array
// to a single string

const names = ["Grace", "Charles", "Ada", "Alan"];

let str = names.join();
// str is "Grace,Charles,Ada,Alan"

str = names.join(", ");
// str is "Grace, Charles, Ada, Alan"

str = names.toString();
// str is "Grace,Charles,Ada,Alan"


// Example 2: Sort numeric values
// in ascending sequence
const numbers = [5, 12, 8, 6, 9, 2];
numbers.sort( (x, y) => x - y );// numbers is [2, 5, 6, 8, 9, 12]
```

Methods Of The Array Type That Transform Elements



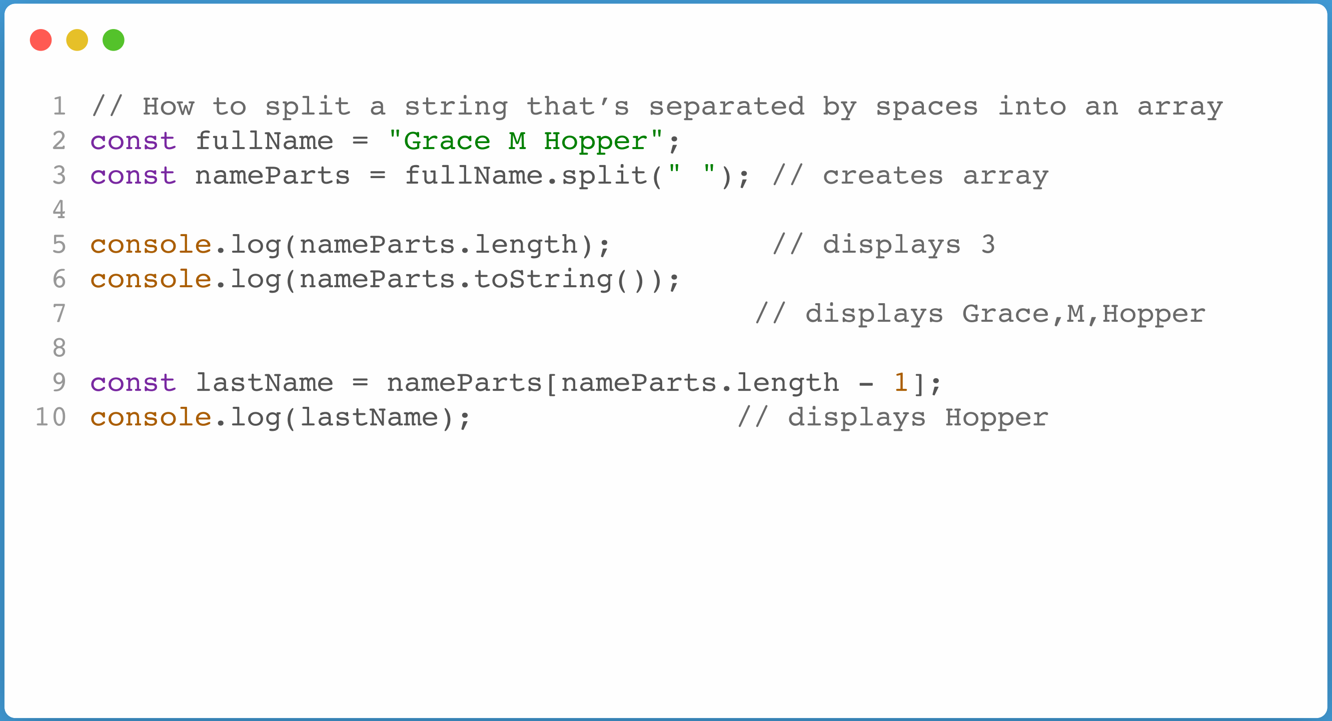
```javascript
// Example 3: Create a new array
// with each element multiplied by 2

const doubled = numbers.map( value => value * 2 );
// doubled is [4, 10, 12, 16, 18, 24]
// numbers is unchanged

// Example 4: Transform the elements
// and then convert to a single string
const names = ["Grace", "Charles", "Ada", "Alan"];
let str = names.reduce( (prev, current) =>
    prev + " " + current.toLowerCase(), "Names:" );
// str is "Names: grace charles ada alan"

// Example 5: Flatten an array that's nested
// two levels deep
const nested = [5, 12, 8, 6, 9, [4, [0, 7], 1], 2];
const flattened = nested.flat(2);
// flattened is [5, 12, 8, 6, 9, 4, 0, 7, 1, 2]
```

More On Arrays

More Skills For Working With Arrays

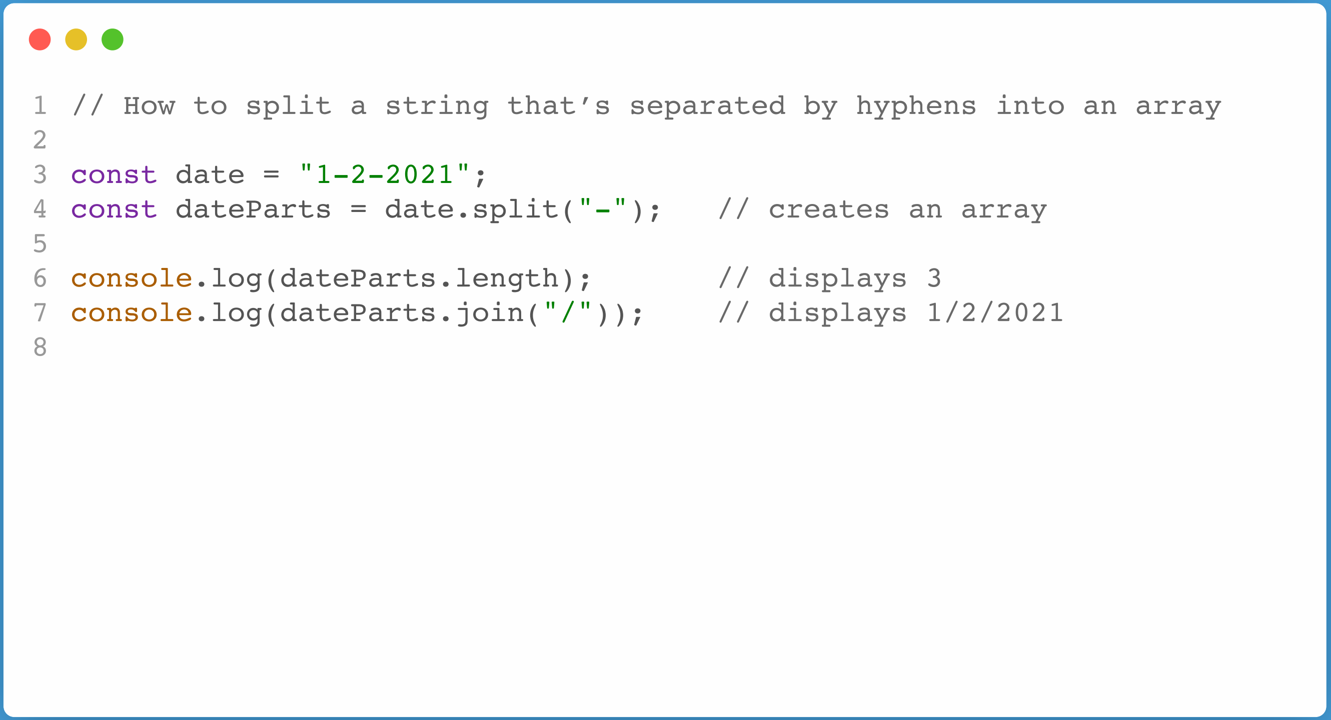
- The *split()* method of a String object can split a string into an array.
- If a string doesn't include the separator specified by the *split()* method, the *split()* method returns the entire string as the first element in a one-element array.
- If the separator specified by the *split()* method is an empty string, the *split()* method returns an array where each character in the string is an element in the array.
- Usage:
  - **split(***separator[, limit]***)**
    - Splits a string into an array based on the value of the separator parameter and returns the array. The optional limit parameter specifies the maximum number of elements in the new array.

More Skills For Working With Arrays



```javascript
// How to split a string that’s separated by spaces into an array
const fullName = "Grace M Hopper";
const nameParts = fullName.split(" "); // creates array

console.log(nameParts.length);         // displays 3
console.log(nameParts.toString());
                                      // displays Grace,M,Hopper

const lastName = nameParts[nameParts.length - 1];
console.log(lastName);               // displays Hopper
```

More Skills For Working With Arrays

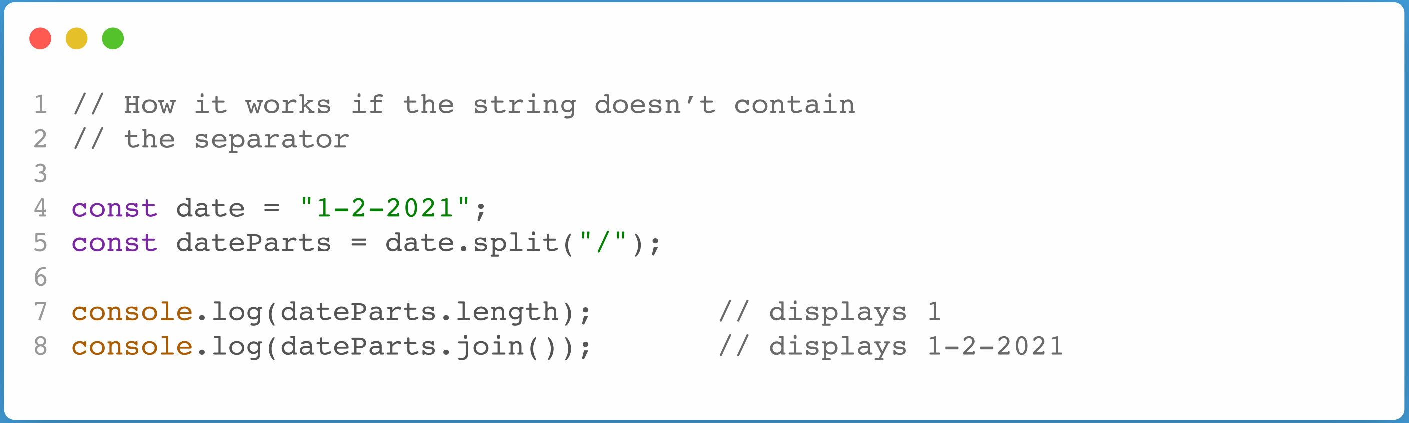


```
// How to split a string that's separated by hyphens into an array

const date = "1-2-2021";
const dateParts = date.split("-");   // creates an array

console.log(dateParts.length);       // displays 3
console.log(dateParts.join("/"));    // displays 1/2/2021

```

More Skills For Working With Arrays

```javascript
// How to split a string into an array of characters

const fullName = "Grace Hopper";
const nameCharacters = fullName.split("");

console.log(nameCharacters.length);  // displays 12
console.log(nameCharacters.join()); // displays G,r,a,c,e, ,H,o,p,p,e,r

```

More Skills For Working With Arrays



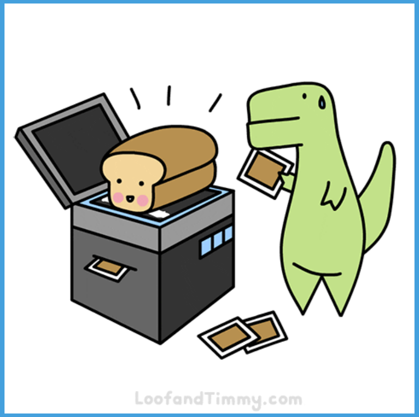
```javascript
// How it works if the string doesn’t contain
// the separator

const date = "1-2-2021";
const dateParts = date.split("/");

console.log(dateParts.length);       // displays 1
console.log(dateParts.join());       // displays 1-2-2021
```

More Skills For Working With Arrays

```javascript
// How to get just one element from a string

const fullName = "Grace M Hopper";
const firstName = fullName.split(" ", 1);

console.log(firstName.length);          // displays 1
console.log(firstName[0]);              // displays Grace
```

Copying Arrays

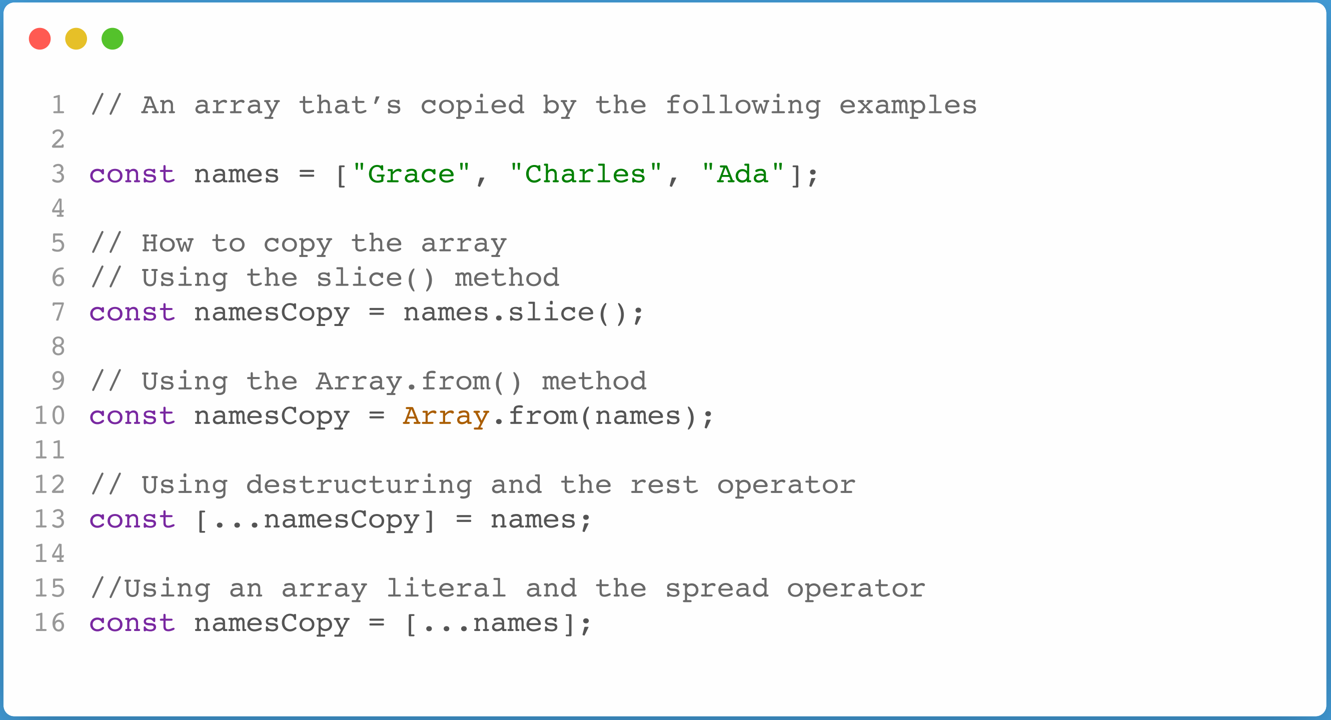
- JavaScript provides several ways to make a copy of an existing array.
- Because the *slice()* method has been available since ECMAScript 1, it will work with older browsers.
- The *Array.from()* method wasn't introduced until ECMAScript 2015, so it doesn't work with older browsers. However, JavaScript libraries called *polyfill* libraries are available that allow the older *Array.from()* method to be used with older browsers.
- You use the *spread operator (. .)* to spread out the elements of an array into a list separated by commas. Then, you can use that list in an array literal that assigns values to the elements of a new array.
- Like the rest operator, the spread operator was introduced with ECMAScript 2015 and can't be used with older browsers.



Copying Arrays

- There are four ways to make a copy of an existing array:
  1. Call the *slice()* method of the array.
  2. Pass the array to the static *from()* method of the Array type.
  3. Destructure the array with the rest operator.
  4. Use an array literal with a spread operator and the array.

Copying Arrays

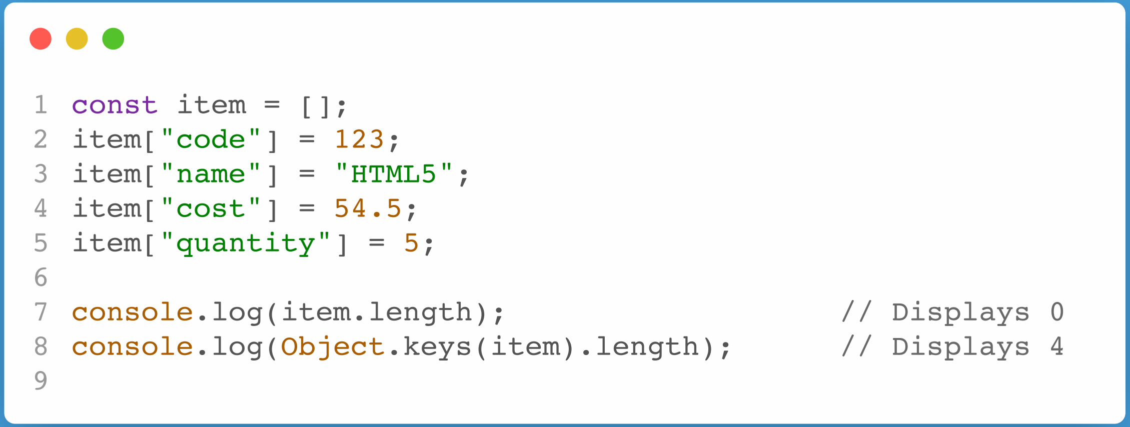


```
// An array that’s copied by the following examples

const names = ["Grace", "Charles", "Ada"];

// How to copy the array
// Using the slice() method
const namesCopy = names.slice();

// Using the Array.from() method
const namesCopy = Array.from(names);

// Using destructuring and the rest operator
const [...namesCopy] = names;

//Using an array literal and the spread operator
const namesCopy = [...names];
```

Associative Arrays

- An *associative array* uses strings as the indexes instead of numbers.
- If you mix numeric and string indexes in an array, the length will indicate only the number of elements with numeric index and a for loop will only process the elements with numeric indexes. However, a for-in loop will process all the elements.
- To get the number of elements with string indexes, you can use the static *keys()* method of the Object type to get an array of all string indexes.
- Because elements with string indexes are actually object properties, some of the array properties and methods don't work as expected with associative arrays. For instance, the *length* property returns 0 and the *pop()* method returns *undefined*.

How To Create An Associative Array With Four Elements

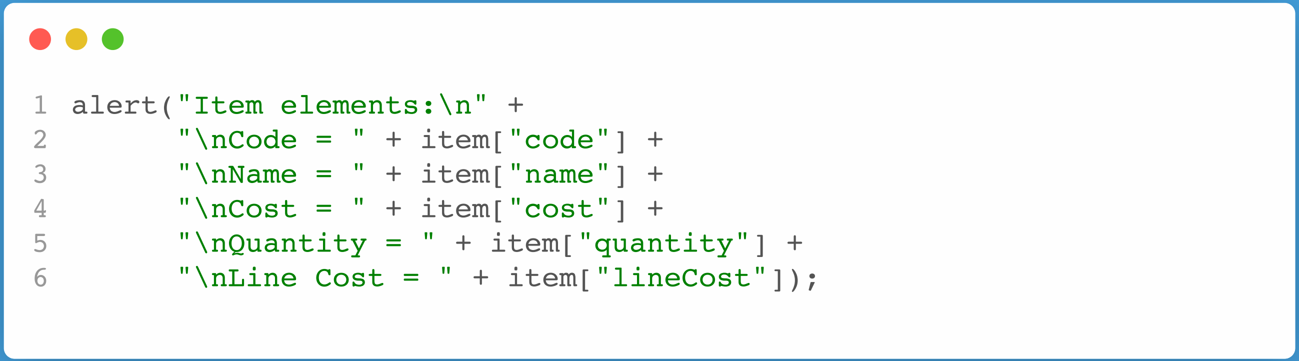


```javascript
const item = [];
item["code"] = 123;
item["name"] = "HTML5";
item["cost"] = 54.5;
item["quantity"] = 5;

console.log(item.length);                     // Displays 0
console.log(Object.keys(item).length);        // Displays 4

```

How To Add An Element To The Associative Array

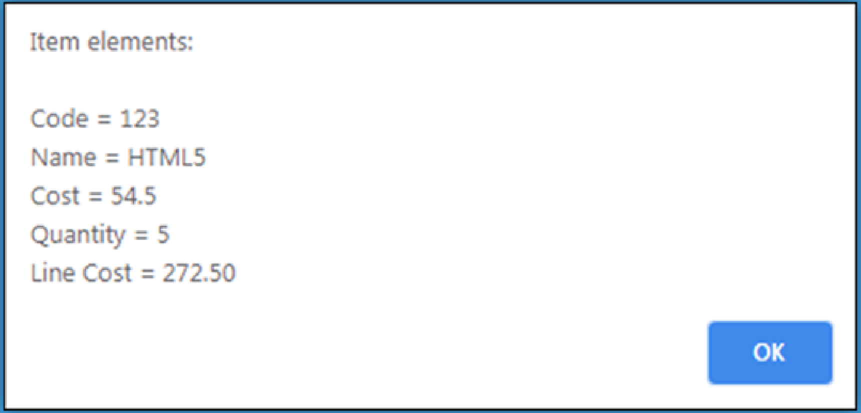
```
item["lineCost"] = (item["cost"] * item["quantity"]).toFixed(2);
```

How To Retrieve And Display The Elements In The Associative Array



```
alert("Item elements:\n" +
      "\nCode = " + item["code"] +
      "\nName = " + item["name"] +
      "\nCost = " + item["cost"] +
      "\nQuantity = " + item["quantity"] +
      "\nLine Cost = " + item["lineCost"]);
```

result . . .



```
Item elements:

Code = 123
Name = HTML5
Cost = 54.5
Quantity = 5
Line Cost = 272.50
```

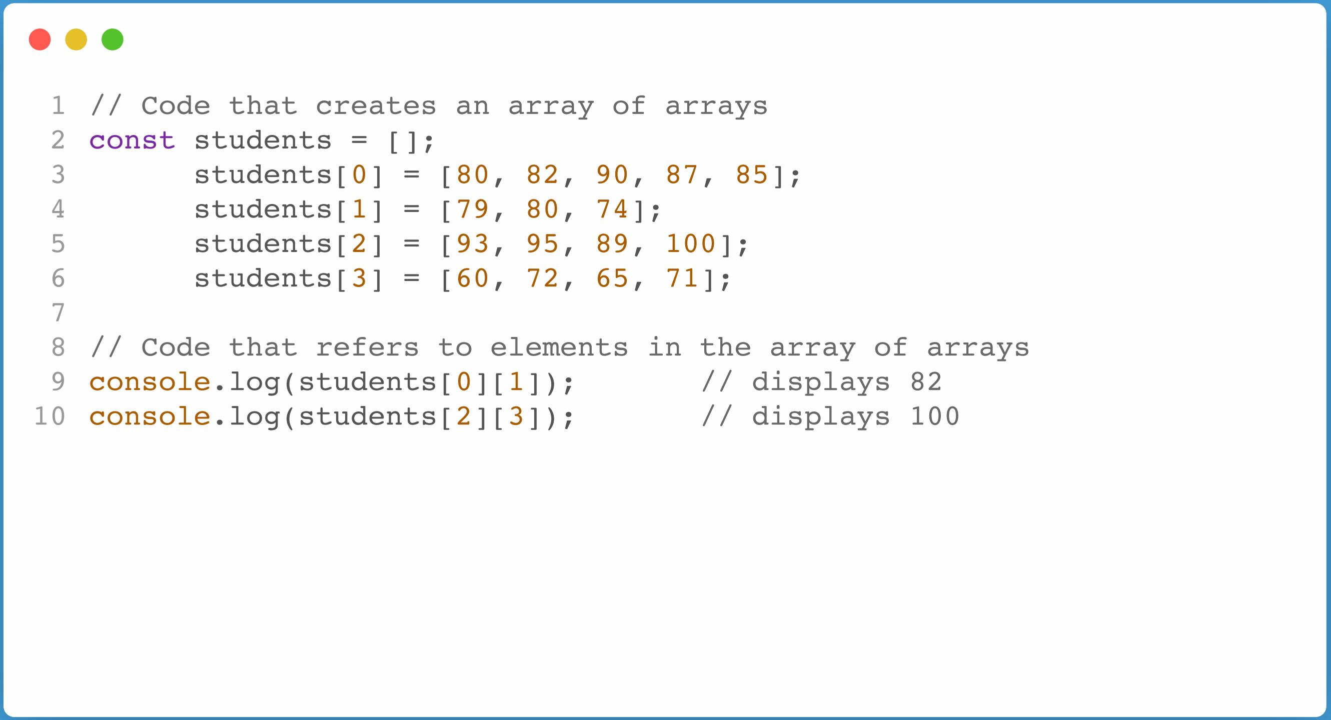
How To Use A For-in Loop With The Associative Array

```
let result = "";
for (let i in item) {
    result += i + "=" + item[i] + " ";
}
// result is "code=123 name=HTML5 cost=54.5 quantity=5
// lineCost=272.50 "
```

Faux Multidimensional Arrays

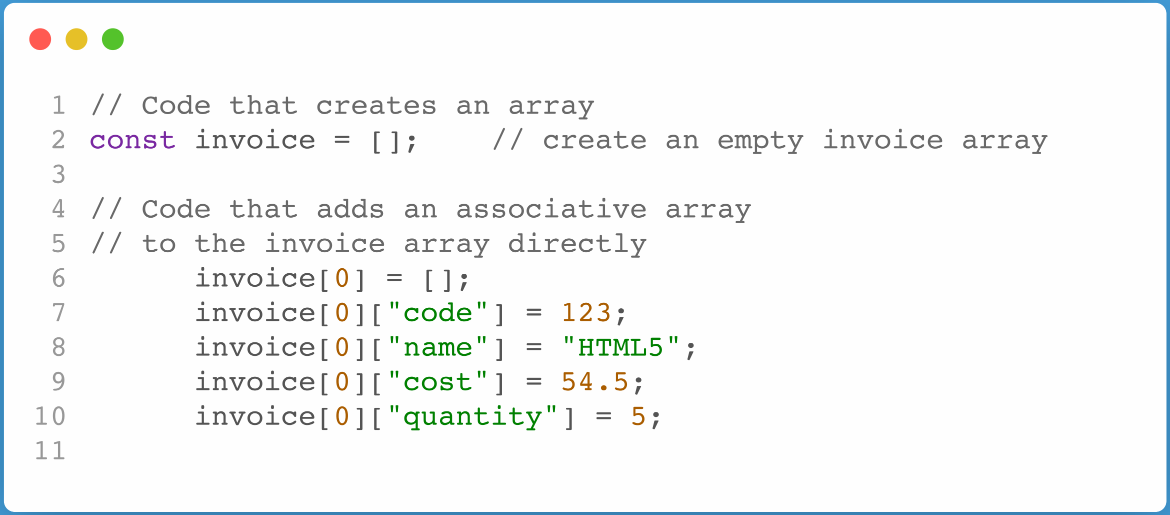
- Although JavaScript doesn't provide for multi-dimensional arrays, you can get the same effect by creating an *array of arrays*.
  - In an array of arrays, each element in the first array is another array.
- The arrays within an array can be regular arrays or associative arrays.
- To refer to the elements in an array of arrays, you use two index values for each element.
  - The first value is for an element in the primary array. The second value is for an element in the nested array.
- If necessary, you can nest arrays beyond the two dimensions that are illustrated in the following slides.
  - In other words, you can create an array of arrays of arrays, and so on. As usual, though, be careful with this as your code can become hard to read and maintain if it's too complex.

How To Create And Use An Array Of Arrays



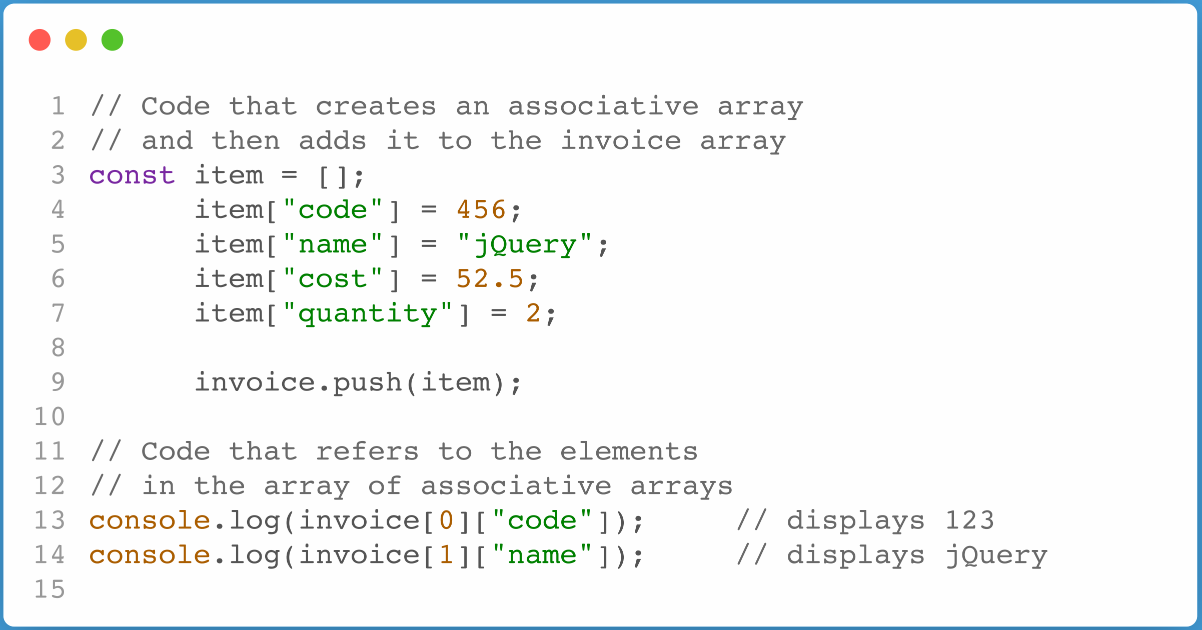
```javascript
// Code that creates an array of arrays
const students = [];
      students[0] = [80, 82, 90, 87, 85];
      students[1] = [79, 80, 74];
      students[2] = [93, 95, 89, 100];
      students[3] = [60, 72, 65, 71];

// Code that refers to elements in the array of arrays
console.log(students[0][1]);       // displays 82
console.log(students[2][3]);       // displays 100
```

How To Create And Use An Array Of Associative Arrays (Part 1)



```javascript
// Code that creates an array
const invoice = [];    // create an empty invoice array

// Code that adds an associative array
// to the invoice array directly
      invoice[0] = [];
      invoice[0]["code"] = 123;
      invoice[0]["name"] = "HTML5";
      invoice[0]["cost"] = 54.5;
      invoice[0]["quantity"] = 5;

```

How To Create And Use An Array Of Associative Arrays (Part 2)

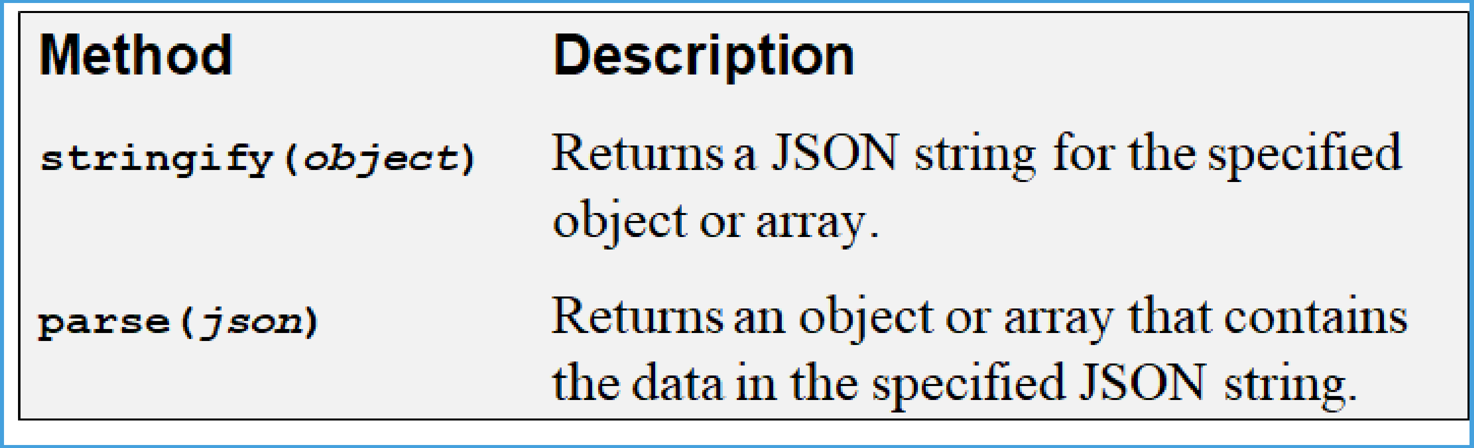


```javascript
// Code that creates an associative array
// and then adds it to the invoice array
const item = [];
      item["code"] = 456;
      item["name"] = "jQuery";
      item["cost"] = 52.5;
      item["quantity"] = 2;

      invoice.push(item);

// Code that refers to the elements
// in the array of associative arrays
console.log(invoice[0]["code"]);     // displays 123
console.log(invoice[1]["name"]);     // displays jQuery

```

Converting Arrays to JSON

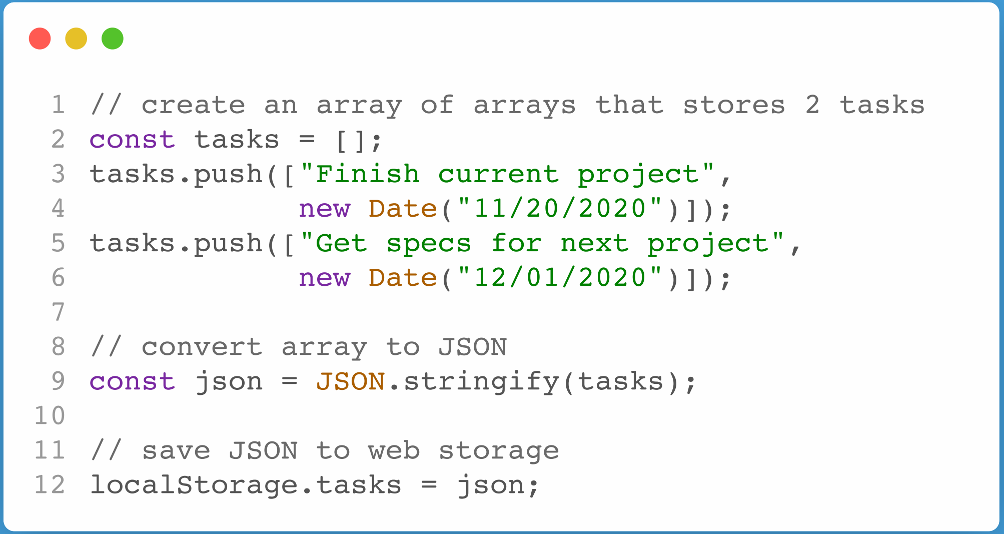
- *JSON (JavaScript Object Notation*) is a format that uses text to store and transmit data for an object such as an array.
- The JSON format was originally based on the notations for objects that was used by JavaScript, but most programming languages now provide a way to generate and parse JSON. As a result, it's a useful data format for sharing object data across languages and applications.
- When using the *JSON.parse()* method to convert a JSON string back into objects, you may need to call the constructor of the object again. For example, to convert a JSON object back to a *Date* object, you need to use the *new* keyword and the *Date()* constructor to create the Date object from the JSON string.
- You can use the *JSON.stringify()* and *JSON.parse()* methods to work with most types of native JavaScript objects as well as user-defined objects.
- JSON often comes into play when working with *AJAX (Asynchronous JavaScript and XML).*
  - *AJAX* will be introduced later in the course.

Two Static Methods Of The JSON Type



Method	Description
`stringify(`*`object`*`)`	Returns a JSON string for the specified object or array.
`parse(`*`json`*`)`	Returns an object or array that contains the data in the specified JSON string.

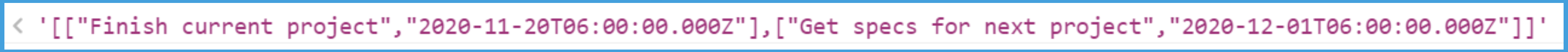
How To Convert An Array Object To JSON



```javascript
// create an array of arrays that stores 2 tasks
const tasks = [];
tasks.push(["Finish current project",
            new Date("11/20/2020")]);
tasks.push(["Get specs for next project",
            new Date("12/01/2020")]);

// convert array to JSON
const json = JSON.stringify(tasks);

// save JSON to web storage
localStorage.tasks = json;
```

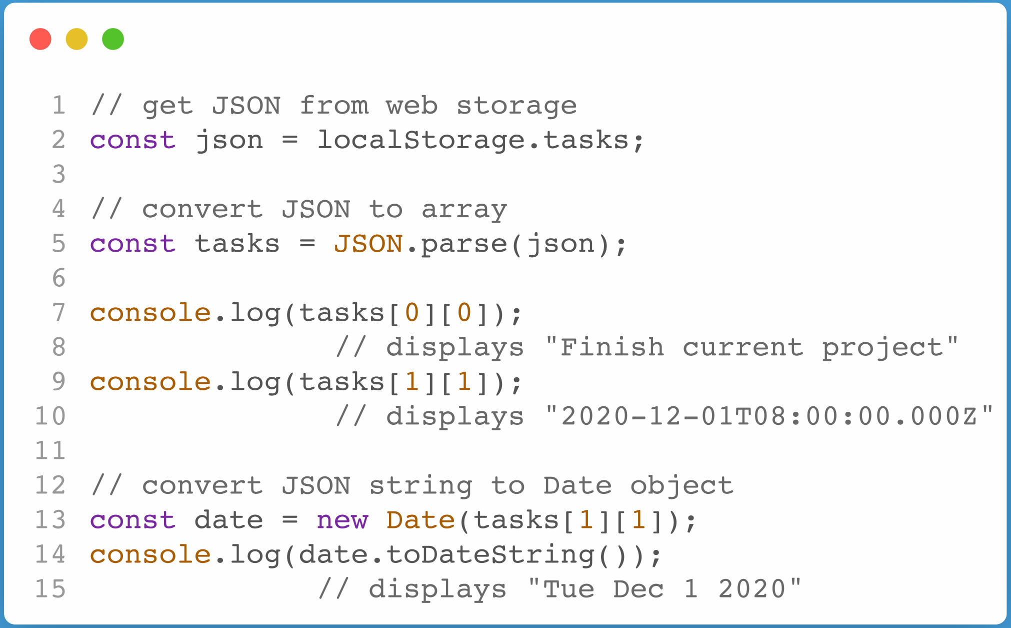
produces the following JSON

Array object



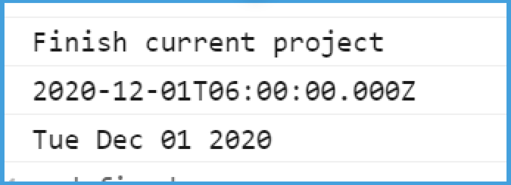
```
< '[["Finish current project","2020-11-20T06:00:00.000Z"],["Get specs for next project","2020-12-01T06:00:00.000Z"]]'
```

How To Convert JSON To An Array Object



```javascript
// get JSON from web storage
const json = localStorage.tasks;

// convert JSON to array
const tasks = JSON.parse(json);

console.log(tasks[0][0]);
                  // displays "Finish current project"
console.log(tasks[1][1]);
                  // displays "2020-12-01T08:00:00.000Z"

// convert JSON string to Date object
const date = new Date(tasks[1][1]);
console.log(date.toDateString());
                 // displays "Tue Dec 1 2020"
```

outputs



```
Finish current project
2020-12-01T06:00:00.000Z
Tue Dec 01 2020
```

Objective 7C

Distinguish between an array, a set, and a map.

Sets

- A *set* can store zero or more elements where each element has a unique value.
    - Arrays can store zero or more elements; however, each value does not have to be unique in an array.
- To create a set, you must use the *Set()* constructor. This constructor accepts an optional array value. If the array contains duplicate values, the constructor removes them when it assigns the array elements to the set.
- A set has less functionality than an array, but you can easily convert a set to an array.
- Why use a set over an array?
    - If we want to avoid duplication.
    - Arrays are indexed collection; sets are *keyed* collections
        - A keyed collection uses keys (or words) instead of numbers to reference elements. Sets can contain numbers, but elements don't have to be referred to by number if the element contains a string.

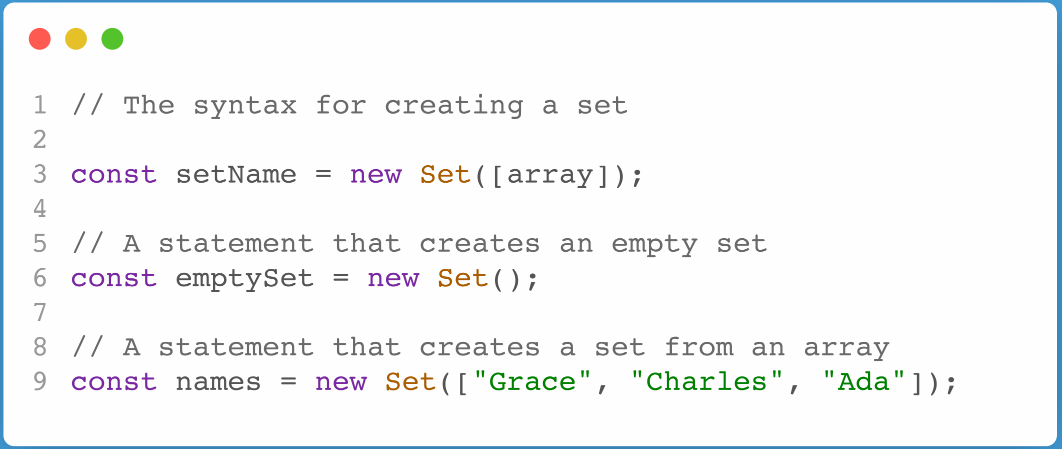
Properties and Methods of the Set Type

- Properties
  - **size**
    - The number of elements in the set (note how this differs from Array's *length*).
- Methods
  - **add(***value***)**
    - Adds the value to the end of the set if it isn't already in the set.
  - **delete(***value***)**
    - Removes the value from the set.
  - **has(***value***)**
    - Returns a Boolean that indicates whether the value is in the set.

Properties and Methods of the Set Type

- Methods (continued)
  - **forEach(*function*)**
    - Calls the function once for each value in the set.
  - **entries()**
    - Returns an iterator with a [key, value] array for each element in the set. For a set, key and value are the same.
  - **values()**
    - Returns an iterator with the value of each element in the set.
  - **keys()**
    - Returns an iterator with the key value of each element in the set.

Basic Sets

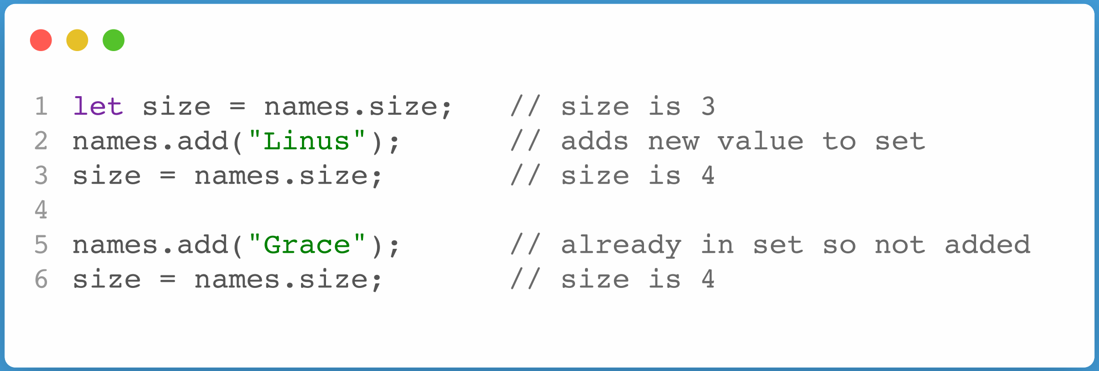


```
// The syntax for creating a set

const setName = new Set([array]);

// A statement that creates an empty set
const emptySet = new Set();

// A statement that creates a set from an array
const names = new Set(["Grace", "Charles", "Ada"]);
```

How To Check For Values In A Set

```javascript
let hasName = names.has("Grace");  // hasName is true
hasName = names.has("Linus");      // hasName is false

```

How To Add Values To A Set



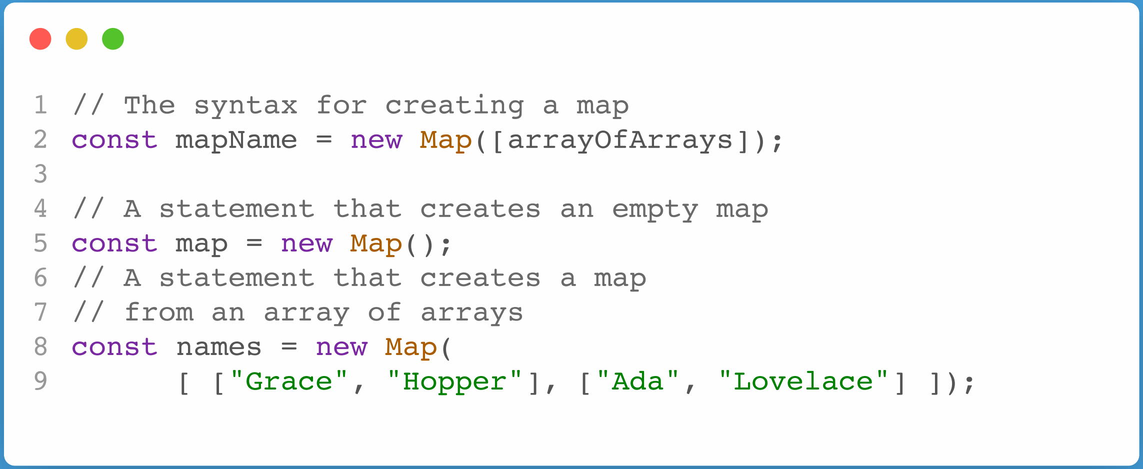
```javascript
let size = names.size;    // size is 3
names.add("Linus");       // adds new value to set
size = names.size;        // size is 4

names.add("Grace");       // already in set so not added
size = names.size;        // size is 4
```

How To Use A Set To Remove Duplicate Values From An Array

```
const dupeArray = [4, 6, 2, 3, 2, 3, 4, 7, 6, 8, 9, 2, 8, 2];
const set = new Set(dupeArray);
const noDupes = Array.from(set);
// noDupes is [4, 6, 2, 3, 7, 8, 9]
```

Maps

- A *map* can store one or more elements consisting of key/value pairs.
  - The *key* for each value must be unique.
  - The *map* "remembers" the original insertion order of the keys.
- To create a map, you must use the *Map()* constructor. This constructor accepts an optional array of arrays. If the nested arrays contain duplicate keys, the constructor replaces the value for each key when it assigns the elements to the map.



```javascript
// The syntax for creating a map
const mapName = new Map([arrayOfArrays]);

// A statement that creates an empty map
const map = new Map();
// A statement that creates a map
// from an array of arrays
const names = new Map(
      [ ["Grace", "Hopper"], ["Ada", "Lovelace"] ]);
```

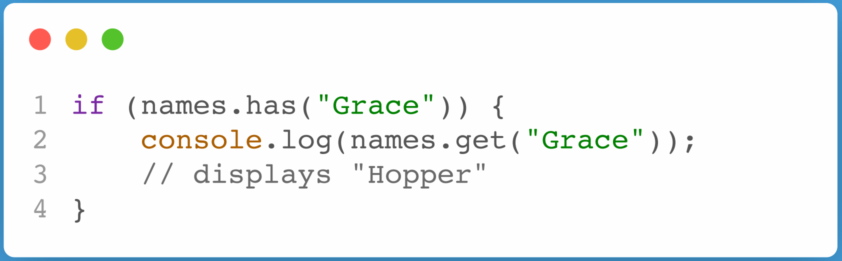
Properties and Methods of the Map Type

- Properties
  - **size**
    - The number of elements in the map (compare to *size* for the equivalent information in the *Set* type, and *length* for the equivalent information in Array).
- Methods
  - **set(***key, value***)**
    - Adds an element to the end of the map if the key isn't already in the map.
  - **get(***key***)**
    - Returns the value associated with the key, or undefined if the key isn't found.
  - **delete(***key***)**
    - Removes the element with the key from the map.

Properties and Methods of the Map Type

- Methods (continues...)
  - **clear()**
    - Removes all the elements from the map.
  - **has(*key*)**
    - Returns a Boolean that indicates whether an element with the key exists.
  - **forEach(*function*)**
    - Calls the function once for each element in the map.
  - **entries()**
    - Returns an iterator with a [key, value] array for each element in the map.
  - **values()**
    - Returns an iterator with the value for each element in the map.
  - **keys()**
    - Returns an iterator with the keys for each element in the map.

How To Check For A Key In The Map And Retrieve Its Associated Value

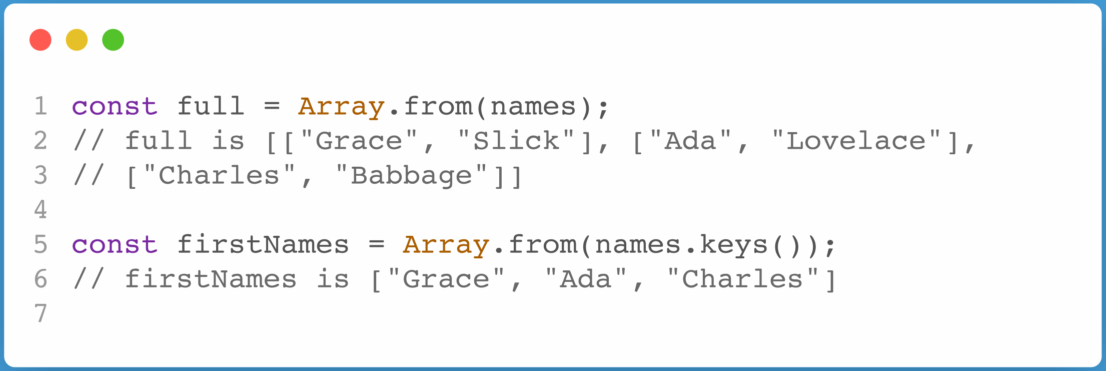


```javascript
if (names.has("Grace")) {
    console.log(names.get("Grace"));
    // displays "Hopper"
}
```

How To Add Key/Value Pairs To The Map

```
let size = names.size;             // size is 2

// add new key/value pair to map
names.set("Charles", "Babbage");
size = names.size;                 // size is 3

// replace value associated with key
names.set("Grace", "Slick");
size = names.size;                 // size is 3
```

How To Get Arrays Of The Keys And Key/Value Pairs Of The Map



```
const full = Array.from(names);
// full is [["Grace", "Slick"], ["Ada", "Lovelace"],
// ["Charles", "Babbage"]]

const firstNames = Array.from(names.keys());
// firstNames is ["Grace", "Ada", "Charles"]

```

Fin

Make sure to review the lecture videos and lab demos before starting your homework. The videos will provide more information on topics discussed in this presentation.