Arrays In JavaScript

Jerry Cain
CS 106AX
October 13, 2023

slides leveraged from those constructed by Eric Roberts
Simple Arrays

• An array is a collection of individual data values in which it is possible to count the values off in order: here is the first, here is the second, and so on.

• The individual values in an array are called elements. The number of elements is called the length of the array. As with strings, you can determine the length of an array by checking its length property.

• Each element is identified by its position within the array, which is called its index.

• In JavaScript, index numbers always begin with 0 and extend up to one less than the length of the array.
Creating an Array

• The simplest way to create an array in JavaScript is to list the elements of the array surrounded by square brackets and separated by commas. For example, the declaration

```javascript
const COIN_VALUES = [1, 5, 10, 25, 50, 100];
```

creates a constant array of six elements that correspond to the standard coins available in the United States.

• Arrays are commonly represented conceptually as a series of numbered boxes, as in the following representation of COIN_VALUES:

<table>
<thead>
<tr>
<th>COIN_VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
Nonnumeric Arrays

- Arrays may contain values of any JavaScript type. For example, the declaration

```javascript
const COIN_NAMES = [
    "penny",
    "nickle",
    "dime",
    "quarter",
    "half-dollar",
    "dollar"
];
```

creates the following array:

<table>
<thead>
<tr>
<th></th>
<th>&quot;penny&quot;</th>
<th>&quot;nickel&quot;</th>
<th>&quot;dime&quot;</th>
<th>&quot;quarter&quot;</th>
<th>&quot;half-dollar&quot;</th>
<th>&quot;dollar&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Array Selection

• Given an array, you can get the value of any element by writing the index of that element in brackets after the array name. This operation is called selection.

• For example, given the declarations on the preceding slides, the value of \texttt{COIN\_VALUES[3]} is 25.

• Similarly, the value of \texttt{COIN\_NAMES[2]} is the string "dime".

\[
\begin{array}{ccccccc}
\text{COIN\_VALUES} & 1 & 5 & 10 & 25 & 50 & 100 \\
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\text{COIN\_NAMES} & "penny" & "nickel" & "dime" & "quarter" & "half-dollar" & "dollar" \\
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]
Cycling through Array Elements

• One of the most useful array idioms is cycling through each of the elements of an array in turn. The standard for loop pattern for doing so looks like this:

```javascript
function sumArray(array) {
  let sum = 0;
  for (let i = 0; i < array.length; i++) {
    sum += array[i];
  }
  return sum;
}
```

• As an example, the following function computes the sum of the elements in array:

```javascript
for (let i = 0; i < array.length; i++) {
  Operations involving the i\textsuperscript{th} element of the array
}
```
Exercise: Making Change

• Write a function `makeChange(change)` that displays the number of coins of each type necessary to produce `change` cents using the values in the constant arrays `COIN_VALUES` and `COIN_NAMES`.

• In writing your program, you may assume that the currency is designed so that the following strategy always produces the correct result:
  – Start with the last element in the array (in this case, dollars) and use as many of those as possible.
  – Move on to the previous element and give as many of those as possible, continuing this process until you reach the number of pennies.

• Assume that someone has written `createRegularPlural`, which is exercise 9 in Chapter 7, on page 266.
# Adding and Removing Elements

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>push(element, . . .)</strong></td>
<td>Adds one or more elements to the end of the array.</td>
</tr>
<tr>
<td><strong>pop()</strong></td>
<td>Removes and returns the last element of the array.</td>
</tr>
<tr>
<td><strong>shift()</strong></td>
<td>Removes and returns the first element of the array.</td>
</tr>
<tr>
<td><strong>unshift(element, . . .)</strong></td>
<td>Adds one or more elements to the front of the array.</td>
</tr>
<tr>
<td><strong>slice(start, finish)</strong></td>
<td>Returns a subarray beginning at <code>start</code> and ending just before <code>finish</code>.</td>
</tr>
<tr>
<td><strong>splice(index, count, . . .)</strong></td>
<td>Removes <code>count</code> elements starting at <code>index</code>, and optionally adds new ones.</td>
</tr>
</tbody>
</table>

Although each of these has its use cases, we’ll initially focus on the three most common operations: **push**, **pop**, and **shift**.
Growing an Array by Accretion

• The `push` method makes it possible to create an array by adding one element at a time. This pattern looks like this:

```javascript
let array = [];
for (whatever limits are appropriate to the application) {
    array.push(the new element);
}
```

• As an example, the following function creates an array of `n` values, each of which is initialized to `value`:

```javascript
function createArray(n, value) {
    let array = [];
    for (let i = 0; i < n; i++) {
        array.push(value);
    }
    return array;
}
```
Passing Arrays as Parameters

• When you pass an array as a parameter to a function or return a function as a result, only the reference to the array is actually passed between the functions.

• The effect of JavaScript's strategy for representing arrays internally is that the elements of an array are effectively shared between the caller and callee. If a function changes an element of an array passed as a parameter, that change will persist after the function returns.

• The next slide simulates a program that does the following:
  1. Generates an array containing the integers 0 to $N-1$.
  2. Prints out the elements in the array.
  3. Reverses the elements in the array.
  4. Prints out the reversed array on the console.
The `reverseArray` Function

```javascript
function TestReverseArray() {
    function reverseArray(array) {
        for (let lh = 0; lh < array.length/2; lh++) {
            let rh = array.length - lh - 1;
            let tmp = array[lh];
            array[lh] = array[rh];
            array[rh] = tmp;
        }
    }
}
```

Array: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Forward: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Reverse: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Console:
Forward: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Reverse: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
# Other Array Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>concat(array, . . .)</code></td>
<td>Concatenates one or more arrays onto the receiver array.</td>
</tr>
<tr>
<td><code>indexOf(element)</code></td>
<td>Returns the first index at which <code>element</code> appears, or –1 if not found.</td>
</tr>
<tr>
<td><code>lastIndexOf(element)</code></td>
<td>Returns the last index at which <code>element</code> appears, or –1 if not found.</td>
</tr>
<tr>
<td><code>reverse()</code></td>
<td>Reverses the elements of the array.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td>Sorts the elements of the array in ascending order.</td>
</tr>
</tbody>
</table>
The End