ARRAYS: DIVERSITY IN UNITY

Lesson Objectives

- Declare and resize JavaScript arrays
- When to use for .. of loops instead of indexed for loops
- How to compare arrays for equality
- Add elements to the beginning and end of arrays
- Use multi-dimensional arrays

Definition

- data structure that can hold multiple elements in consecutive memory locations
- Memory locations are indexed
  - 0 is index for the first element
- In JavaScript, dynamic in both length and types of elements it can hold
  - Similar to ArrayList in Java (dynamic length)
  - Best practice to only hold one type of element

Declaring an Array

- Using array literal syntax

```
const numbers = [];
const fruits = ["Apple", "Banana", "Mango"];
```

- can also be created using new keyword

```
const numbers = new Array(6);
```

- generally, use literal syntax

Using an Array

- Indices start with zero.
- get an element by its number in square brackets:

```
let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

- replace an element

```
fruits[2] = 'Pear'; // now ["Apple", "Orange", "Pear"]
```

- add a new one

```
fruits[3] = 'Lemon'; // now ["Apple", "Orange", "Pear", "Lemon"]
```

Size of an array

- built-in property, `length` represents current size of array
- total count of elements in array is its `length`

```
let numbers = []
console.log(numbers.length); // 0
numbers = [1,2,3];
console.log(numbers.length) // 3
```

A word about “length”

- `length` property automatically updates when modify the array.
  - not the count of values in the array, but the greatest numeric index plus one.
  - For instance, a single element with a large index gives a big length:

```
let fruits = [];
fruits[123] = "Apple";

alert( fruits.length ); // 124
```

- usually don’t use arrays like that.

Filling an Array

- Loops can be used to fill an array with some default values, usually for testing purposes.

```
const scores = [];
for (let i=0; i<10; i++){
        scores[i] = Math.ceil(Math.random()*100);
}

console.log(scores);
```

Exercise

- Write code to create an array named scores and fill it with 5 test scores 10, 20, 30, 40 and 50.
- Now write a function named findAverage, that takes an array as an argument and return average of the array values.
- Call findAverage function passing array you created in step1 and save the return result in a variable, average.
- Print the average, it should be 30 for this example.
- Create a second array filled with 10 random values between 0 to 10 and find the average of the array values.
- Should compute correct average for an array of any size.

Main Point

Using an array, we can hold many elements under a single identifier, which eliminates the need for unique identifiers for every value. *Science of consciousness, during transcendence our bounded individual identity identifies with the unbounded cosmic identity*.

Looping through an array

- Traditional way to cycle array items is the for loop over indexes:

```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

- But for arrays there is another form of loop, `for..of`:

```
for (let fruit of fruits) {
  alert( fruit );
}
```

- The for..of doesn't give access to the index of the current element, just its value, but in most cases that's enough.
  - And it's shorter.
  - And avoids bugs that often occur from index errors at the end points
  - Favor for..of as default loop over arrays unless really need index

Array comparison

- Arrays are type Object
- When == or === operators are used on JavaScript objects, their references are compared

- **If array comparison is needed compare them item-by-item in a loop**.
  - Mocha has a very convenient assert.deepStrictEqual
  - [1, 2, 3] === [1, 2, 3] → false

Array methods

- JavaScript provides several useful methods that one can use to manipulate contents of an array.
  - add/remove array contents to/from beginning and end of an array.
  - run a function on every element of the array.
  - sort and search
  - split and join and so on ...

`toString`

- Arrays have their own implementation of `toString` method that returns a comma-separated list of elements.
- For instance:

```
let arr = [1, 2, 3];

console.log( arr ); // [1,2,3]
console.log( arr.toString() === '1,2,3' ); // true
```

Add/Remove elements To/From the beginning

- built in methods to add/remove elements to/from beginning of array.
  - `shift`: extracts the first element of the array and returns it:

```
let fruits = ["Apple", "Orange", "Pear"];
console.log( fruits.shift() ); // remove Apple and alert it
console.log( fruits ); // Orange, Pear
```

  - `unshift`: add the element to the beginning of the array

```
let fruits = ["Orange", "Pear"];
fruits.unshift('Apple');
console.log( fruits ); // Apple, Orange, Pear
```

Add/Remove elements To/From the end

- add/remove elements to/from the end of the array.
  - `pop`: extracts the last element of the array and return it.

```
let fruits = ["Apple", "Orange", "Pear"];
console.log( fruits.pop() ); // remove "Pear" and log it
console.log( fruits ); // Apple, Orange
```

  - `push`: append element to the end of the array.

```
let fruits = ["Apple", "Orange"];
fruits.push("Pear");
console.log( fruits ); // Apple, Orange, Pear
```

  - The call `fruits.push("Peach")` is equal to `fruits[fruits.length] = "Peach"`

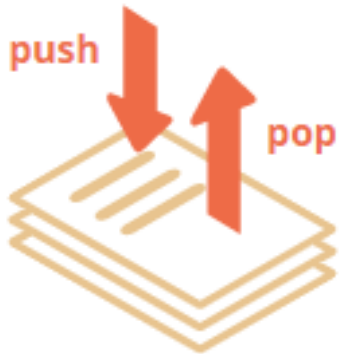
Array as a queue

- A queue is a common use of an array.
  - In computer science, this means an ordered collection of elements which supports two operations:
    - `push` (enqueue) appends an element to the end
    - `shift` (dequeue) get an element from the beginning, advancing the queue, so that 2nd element becomes the 1st.

  - For queues, we have FIFO (First-In-First-Out) principle.
  - Practical applications are common
    - For example, a queue of messages that need to be shown on-screen.

Array as a stack

- There’s another use case for arrays – the data structure named stack.
- It supports two operations:
  - push adds an element to the end.
  - pop takes an element from the end.
- So new elements are added or taken always from the “end”.



- For stacks, the latest pushed item is received first, that’s also called LIFO (Last-In-First-Out) principle

Exercises

Given an expression array exp, write a program to examine whether the pairs and the of “{“, “}” are balanced in exp.

- Use a for .. of loop through the expression array
- push any right bracket onto a stack
- on a left bracket pop the stack and check that return value is a right bracket
- if not, then not balanced
- if stack empty at end then balanced, else not balanced

Example:
Input: exp = ["{", "}", "{", "{", "}", "}“]
Output: Balanced

Input: exp = ["{", "{", "}", "{"]
Output: Not Balanced

Main Point

If you do not need the array index when looping through array elements, the for..of loop is simpler and more error-free. Push/pop, shift/unshift, toString are convenience methods on Array to simplify common operations on arrays. *Science of consciousness, nature always takes the path of least action.*

Multidimensional arrays

- Arrays can have items that are also arrays.
  - We can use it for multidimensional arrays, for example to store matrices:

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
  ];

  console.log( matrix[1][1] ); // 5, the central elemen
t
```

Accessing elements

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]];

console.log(matrix);

for (let i = 0; i < matrix.length; i++) {
    for (let j = 0; j < matrix[i].length; j++) {
        console.log(matrix[i][j]);
    }
}
```

Exercise

- Write a function that accepts a two-dimensional array of numbers and returns the sum of all the elements in the array.

Main Point

In JavaScript multidimensional arrays are arrays of arrays. *Science of consciousness, building a more complex data structure by repeated use of a simpler structure is an example of a common phenomena in nature that complex things are generally combinations of simpler things. Ultimately, everything starts with self referral awareness, awareness being aware of itself.*

View Mocha results in browser

- Mocha results can be conveniently viewed in a browser
- Very nice for teammates, your portfolio, and getting feedback from professor

- Simple demo of Mocha test in a browser – everything in one file

- it is better to have the JavaScript code separate from the html file

- GitHub has a convenient free web server that will allow us to put web pages on the Internet
  - see Sakai > Resources > lab helpers > githubPagesHosting.pdf to create your own GitHub pages website
  - now your Mocha tests will be easily viewable to anyone on the Internet

- develop in node.js environment, then put into browser environment when finished
  - Must use CommonJS modules for node environment, but remove in browser environment
  - See demo code and skeleton code for today's assignment