CTD Intro Week 4

JavaScript Array Methods

Review of Array Basics

- A collection of any type of object indexed by integers
  - Elements are accessed using []
  - First element is 0 – *someArray*[0]
  - Last element is *someArray*[*someArray*.length – 1]
  - .length is an attribute which is one greater than the index of the last element
- Create them using array literals
  - const *myArr* = []; // an empty array
  - Can also use new Array(*size*) // *size* undefined elements
- Accessing an out-of-range element returns undefined

An Assortment of Array Methods

- Just a subset, lots more here: JavaScript Array Methods (w3schools.com)
- *myArr*.sort(*compare-function*) // modifies (sorts) the array
  - *compare-function* returns (negative, 0, or positive) for any pair of elements
  - The *compare-function* is optional and ascending order is the default
- *myArr*.slice([*startElt*[, *endElt*]]) // returns a new array, does not modify
  - [, *endElt*] is a way to document an optional argument
  - Takes the elements with index *startElt* up to but not including *endElt*.
    - To the end of the array is *endElt* is not specified
- *myArr*.splice(*position*, *number-to-remove*, [*newElt*[,…*newElt*]])
  - [*newElt*[,…*newElt*]] is a way of saying that 0 or more *newElt's* can be specified
  - Modifies *myArr* starting at *position*
  - Removes *num-to-remove* elements and returns them in an array (can be zero)
  - Replaces the removed elements with the *newElts*, if there are none, there is no replacement

```
// example of sort
let sorted = nums.sort((a, b) => b - a);
// nums was [1, 2, 3, 4, 5, 6, 7, 8]
// nums and sorted are now [8, 7, 6, 5, 4, 3, 2, 1]

// example of slice
let sliced = nums.slice(1, 3);
// sliced is [2, 3]
// nums is still [1, 2, 3, 4, 5, 6, 7, 8]

// example of splice
let removed = nums.splice(1, 2, 'new', 'stuff');
// nums is now [1, 'new', 'stuff', 4, 5, 6, 7, 8]
// removed is now [2, 3]
```

More Array Methods

- *myArr*.toString() // converts the array to a single string with commas separating the values
- *myArr*.push(*newThing*) // modifies the array, adds a new last element
  - Returns the new length
- myArr.pop() // modifies the array, removes the last element
  - Returns the removed element
- *myArr*.unshift(*newThing*) // same as push but at the beginning
- *myArr*.shift() // same as pop but at the beginning
- Push/pop/shift/unshift can be used for data structures such as:
  - stacks (last-in/first-out: LIFO), queues (first-in/first-out: FIFO)
- How about a demo?

```
// example of toString
let csLine = nums.toString()
// nums is still [1, 2, 3, 4, 5, 6, 7, 8]
// csLine is now '1,2,3,4,5,6,7,8'

// example of push
let newCount = nums.push(63);
// nums is now [1, 2, 3, 4, 5, 6, 7, 8, 63]
// newCount is now 9

// example of pop
let popped = nums.pop();
// nums is now [1, 2, 3, 4, 5, 6, 7, 8]
// popped is now 63
```

Higher Order Functions (Functional Programming)

- Several of this week's lessons use higher order functions
- let *newArray* = *arrayVar*.map(*function*);
  - *function* is any function which takes 1 variable
  - *newArray* is a new array which contains the result of running *function* on each value in *arrayVar.*
- let *filteredArray* = *arrayVar.*filter(*predicate-function*);
  - *predicate-function* is a function of 1 variable which returns true or false
  - *filteredArray* is a new array which contains only the values of *arrayVar* for which *predicate-function* returns true
- let *reducedArray* = *arrayVar*.reduce(*combiner-function*);
  - *combiner-function* is a function of two variables:
    - accumulator
    - currentValue
  - *reducedArray* is a single value which is the result applying *combiner-function* to each element of the array, combining the *currentValue* with the *accumulator*
    - The *combiner-function* and reduce have other optional arguments which are not needed for this exercise

```
// an example of map
let nums = [1, 2, 3, 4, 5, 6, 7, 8];
let unsquared = nums.map((num) => Math.sqrt(num));
// nums is still [1, 2, 3, 4, 5, 6, 7, 8]
// unsquared is:
// [1, 1.414, 1.732, 2, 2.236, 2.449, 2.645,2.828]

// an example of filter
let div3 = nums.filter((num) => num % 3 == 0);
// div3 is [3, 6]
// nums is still [1, 2, 3, 4, 5, 6, 7, 8]

// example of reduce
let factorial = nums.reduce((acc, cur) => acc * cur);
// factorial is 40320
// nums is still [1, 2, 3, 4, 5, 6, 7, 8]
// as it runs:
// 1 * 2 [3, 4, 5, 6, 7, 8] acc: 1, cur: 2
// 2 * 3 [4, 5, 6, 7, 8] acc: 2, cur: 3
// 6 * 4 [5, 6, 7, 8] acc: 6, cur: 4
// 24 * 5 [6, 7, 8] acc:24, cur: 5
// 120 * 6 [7, 8] acc: 120, cur: 6
// 720 * 7 [8] acc: 720, cur: 7
// 5040 * 8 acc: 5040, cur: 8, result: 40320
```

More Higher Order Functions

- *arrayVar*.forEach(*function*);
  - *function* is any function which takes 1 variable
  - The forEach method always returns undefined.
  - forEach is used to produce side-effects.
- let *firstTrue* = *arrayVar.*find(*predicate-function*);
  - *predicate-function* is a function of 1 variable which returns true or false.
  - *firstTrue* is the first element of *arrayVar* for which *predicate-function* returns true.

```
// an example of forEach
let nums = [1, 2, 3, 4];
// returns undefined
nums.forEach((num) => console.log(num * num));
// nums is still [1, 2, 3, 4]
// Console output:
// 1
// 4
// 9
// 16

// example of find
let ints = [1, 3, 5, 7, 8, 9, 10];
let firstEven = ints.find((int) => (int % 2) === 0);
// ints is still [1, 3, 5, 7, 8, 9, 10]
// firstEven is 8
```

Q & A and Demo