Basics of Javascript Arrays

Relevant Links

- Flanagan's book, 7.1-7.8
- MDN's page on the Array global[1]

Javascript Arrays

- In Javascript, arrays can contain absolutely any elements and have a variable length. In this way they are very much like Python's lists.
- Easiest way to create is with an array literal, e.g. `[]`, `[1, 2, 3]` or `[1, [3, 4]]`.
- Access an array value via bracket notation: `arr[2]`. Indexing starts at 0.
  - Question: How would we access the value 4 in the example above?
- Set any array value similarly: `arr[5] = 2`. You can set values out of bounds!
- Arrays are actually just objects, and can have properties that are non-numeric.
- The length of an array is one more than the largest numeric property.

```
let a = [1, 4, 5];
a[2];                    // 2 -> 5
a[6] = 2;
a;
a.length;                // 7
a.foo = 5;               // A random property.
```

- The most basic way to iterate over an array's elements is with a `for` loop:

```
let a = [1,2,3,4];
for (let i = 0; i < a.length; i += 1) {
   console.log(a[i]);
}
```

- A better way to iterate over an array, or in fact any *iterable* object (we will discuss those later), with a `for-of` loop, which is similar to Python loops:

```
let arr = [3,5,3,4];
for (let x of arr) {
   console.log(x);
}
// Can use const if you don't try to reassign it in the loop
for (const x of arr) {
   console.log(x);
}
```

Practice: Create an array containing the squares of the numbers from 1 to 10. Then write a loop that prints them.

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Standard Methods

Consult individual method pages as well as section 7.8 from the book.

Array.from[2] Used to construct an array from an iterable object.
Array.of[3] Used to construct an array from the provided arguments. You should prefer this (or the array literal notation) to using the constructor.

Iterators:

keys[4] Iterates over the indices of the array: `for (const i of [2,3,5].keys()) { console.log(i); }`
values[5] Iterates over the values of the array: `for (const e of [2,3,5].values()) { console.log(e); }`
entries[6] Allows you to iterate over both indices and values of an array: `for (const [i, e] of [2,3,5].ent`

Inserting/Removing elements:

push[7] adds one or more elements to the end of the array. *Returns the new length of the array.*
pop[8] removes the last element of the array and returns it.
unshift[9] adds one or more elements at the beginning of the array, shifting other elements to the right. *Returns the new length of the array.*
shift[10] removes the first element of the array and returns it. Shifts all other elements accordingly.

Slicing:

slice[11] returns a *new* array containing a specific range of elements from the original array.
splice[12] removes and/or inserts elements at a specified location in the array.

Finding:

indexOf[13] searches into an array looking for a specific element. Returns the index of the first match, or -1 if the search fails.
lastIndexOf[14] finds the last match instead.
findIndex[15] searches into an array looking for a specific element that satisfies a provided predicate function.

Others:

reverse[16] reverses the array *in place*.
sort[17] sorts the array *in place*. You can provide a custom sorting function, a topic we will discuss more later.
concat[18] returns a new array comprising of the concatenation of the original array and the arguments.
join[19] used for arrays of strings. Join the strings together, possibly inserting a separator.

There is another set of methods following a higher-order-function paradigm. We will discuss these in future segments.