Part I: Array Processing

Topic Overview
- Modifying Arrays
- The `map` function
- The `reduce` function
- The `filter` function

Modifying Arrays I
- Several functions support modifying arrays
- Adding/removing elements

Modifying Arrays II
- From the end of the array:
  - `arr.push(x)` – adds `x` to the end of the array
  - `y = arr.pop()` – removes and returns the last element in the array
  - Can treat an array as a stack (LIFO)

Modifying Arrays III
- From the start of the array:
  - `arr.unshift(x)` – adds `x` to the start of the array
  - `y = arr.shift()` – removes and returns the first element in the array
  - Can treat an array as a queue (FIFO) using combination of `push` and `shift`
Modifying Arrays IV

More general:

`arr.splice(index, numRemove, [item1, item2, ..., itemN])`

- Allows you to add/remove any number of elements from anywhere in the array
- `index` – index where the add/remove operation occurs in the array
- `numRemove` – number of elements to remove (zero for adding)
- `[item1, item2, ..., itemN]` – an optional list of elements to add (omit for removal)

Demonstration

Map Function

- The map function takes a callback that is applied to each element
- The callback operates on each element and returns a new value
- The result is a new array containing the returned values

```javascript
var arr = [10, 20, 30, 40];
var newArr = arr.map(function(value, index, array) {
  return value * 2;
});
// newArr is now [20, 40, 60, 80]
```

Reduce Function

- The reduce function iterates over elements and aggregates them
- Takes a callback that has a `previousValue` and `currentValue` (in the array)
- Returned value is carried over into the next callback call
- Initial value can also be provided

```javascript
// flatten an array of values into a table row:
var arr = ["Margaret", "Hamilton", "1936/09/17"];
var row = arr.reduce(function(pVal, cVal, index, array) {
  return pVal + `<td>${cVal}</td>`;
}, `<tr>`) + `</tr>`;
```

Filter Function

- The filter function iterates over elements and either retains or ignores them
- Takes a callback that returns `true` (retain) or `false` (ignore)
- Creates a new array of retained values

```javascript
var arr = [5, 8, -2, 0, 4, -4, 8];
// filter only positive values:
var newArr = arr.filter(function(value, index, array) {
  return value > 0;
});
// newArr is now [5, 8, 4, 8]
```

Part II: String Processing

Topic Overview

- Immutability
- Simple modifying functions
- Other functions:
  - `charAt()`
  - `search()`
  - `substr()`
  - `substring()`
  - `split()`
- Polyfills & Shims
Immutability

- Strings in JavaScript are immutable
- Once created, they cannot be changed or modified
- Only way to "modify" a string is to create a new one
- Various functions are supported to create a new, modified string

Simple Examples

- `s.trim()` returns a new string of `s` with leading and trailing whitespace removed
- `s.toUpperCase()` returns a new string of `s` with all alphabetic characters changed to upper case
- `s.toLowerCase()` returns a new string of `s` with all alphabetic characters changed to lower case
- `s.replace()` returns a new string of `s` with certain characters/sequences replaced with others (requires knowledge of regular expressions)

Simple Examples

```javascript
var s = " Hello World! \t ";
var x;

x = s.trim(); //s remains " Hello World! \t 

x = s.toUpperCase(); //HELLO!

x = s.toLowerCase(); //hello!
```

charAt()

- `s.charAt(i)` returns the character of `s` at index `i`
- Like arrays, strings are index from 0

```javascript
var s = "Hello World!";
var x;

x = s.charAt(0); //"H"

x = s.charAt(5); //" 

x = s.charAt(11); //"!

x = s.charAt(12); //"" (empty string)
```

search()

- `s.search(a)` – searches the string `s` for the string `a` and returns the index at which it finds it
- Returns -1 if `s` does not contain `a`

```javascript
var s = "Hello World!";
var x;

x = s.search("World"); //6

x = s.search("o"); //4

x = s.search("zzz"); //-1
```

substr()

- `s.substr(start, length)` – returns a substring of `s` starting at the `start` index and of length equal to `length`
- `length` can be omitted, which returns the rest of the string

```javascript
var s = "Software Development";
var x;

x = s.substr(0, 8); //"Software"

x = s.substr(9); //"Development";

x = s.substr(0, 4); //"Soft"
```
**substring()**

- `s.substring(start, end)` – returns a substring of `s` starting at the `start` index and ending at `end-1`
- Total length of the new string is `end - start`

```
var s = "Software Development";
var x;
x = s.substring(4, 8); //"ware"
x = s.substring(9, 12); //"Dev";
x = s.substring(0, 4); //"Soft"
```

**split()**

- `s.split(delimiter)` – returns an array of strings of `s` broken up along the provided `delimiter`
- `delimiter` character is not included in result(s)

```
var s = "Grace,Hopper,1906-12-09,Rear Admiral,Navy";
x = s.split(","); //[ 'Grace', 'Hopper', '1906-12-09', 'Rear Admiral', 'Navy' ]
x = s.split("0"); //[ 'Grace,Hopper,19', '6-12-', '9,Rear Admiral,Navy' ]
x = s.split("o"); //[ 'Grace,H', 'pper,1906-12-09,Rear Admiral,Navy' ]
```

**Polyfills & Shims I**

- Common operation: search an array
- ES5.1 and prior: we only have `indexOf(x)`
- Very limited: returns index of first string/number matching `x`
- Newer versions (ES6) support array `find` function
- Much better: can search arrays of any types (not just strings/number), takes a callback for more complex logic

**Polyfills & Shims II**

- Problem: want to use this new functionality, but we can’t assume all our users will have modern browsers/support ES6
- Solution: use a polyfill
  - Write code to check if the `find` function is available
  - If it is, do nothing, our code will use the “native” support
  - If it does not, add the functionality with our own code
- “Future” proofs our code, makes our code cross-browser compatible
- Related: a shim augments or changes standard behavior