Notes on Arrays

Ways to declare an array

```
[]  // empty array
[0, 1, 2] // holding some values
[23, 'hello', false, [1, 2], [true, 'hello']] // holding any other object

var count = new Array(1, 2, 3) // using the `Array` constructor
var count = [1, 2, 3] // the above, but more commonly seen
```

Ways to interact with an array

- Count the length

```
var count = [1, 2, 3];
count.length;
```

- Run code for each element

```
var count = [1, 2, 3, 4]
var i;
for (i = 0; i < count.length; i += 1) {
  console.log(i); // or to access the content of the array, `console.log(count[i];)`
}
```

Accessing, modifying, detecting values

```
var count = [1, 2, 3]
count[5] = [1, 2, 3, undefined, undefined, 5]; // the skipped values are undefined
count.length // this is 6
count.length = 4 // we can reassign the length
count; //[1, 2, 3, undefined], the rest are truncated
```

Arrays as objects

```
typeof [];  // object
Array.isArray([]);        // true
Array.isArray('array');   // false
```

Array operations

- JS has an `Array` global object, it has a prototype object
- The prototype object defines the methods for Arrays
- All JS arrays inherit from the prototype object
- Some of the most commonly used `Array` methods are: `push`, `pop`, `unshift`, `shift`, `indexOf`, `lastIndexOf`, `slice`, `splice`, `concat`, `join`

Arrays and operators

- Operators are not useful with Array objects
- But it does not throw an error, and can hide bugs that would otherwise be apparent in other languages

```
// adding array and strings would just coerse array into a string
var initials = ['A', 'B', 'C'];
initials + 'B'; // 'A,B, CB'
initials; //['A', 'B', 'C']

// similarly, explicitly coersing it, although the op is non-mutating
String([initials]); // 'A,H,E'

// adding two arrays also won't work
initials + ['B']; // 'A, H, EB'

// * also does not work in concatenation
[1] * 2; // 2 => '1' * 2 => 1 * 2
[1, 2] * 2; // NaN => '1,2' * 2, then NaN * 2

// other operators also coerse array into string
[5] - 2;              // 3
[5] - [2];            // 3
5 - [2];              // 3
5 - [2, 3];           // NaN -- becomes 5 - '2,3', then 5 - NaN
[] + [];              // '' -- becomes '' + ''
[] - [];              // 0 -- becomes '' - '', then 0 - 0
+[];                  // 0
-[];                  // -0
```

- Comparison operators are also surprising

```
var friends = ['Bob', 'Josie', 'Sam'];
var enemies = ['Bob', 'Josie', 'Sam'];
friends == enemies;                    // false
friends === enemies;                   // false
[] == [];                              // false
[] === [];                             // false
```

- Two arrays are unequal because they are different objects, even though they contain the same values
- Equality operator checks if two arrays are the same array, not if they contain the same content
- They are only equal when they are the same object, i.e. comparing one to itself

```
var friends = ['Bob', 'Josie', 'Sam'];
friends == friends;                    // true
friends === friends;                   // true
```

- If assigning the same array to another variable, they are recognized as the same array

```
var friends = ['Bob', 'Josie', 'Sam'];
var roommates = friends;
friends == roommates;                  // true
friends === roommates;                 // true
```

- When an array is compared with a non-array using the non-strict equality operator, JS coerces array into string before performing the comparison

```
[] == '0';                // false -- becomes '' == '0'
[] == 0;                  // true -- becomes '' == 0, then 0 == 0
[] == false;              // true -- becomes '' == false, then 0 == 0
[] == ![];                // true -- same as above
[null] == '';             // true -- becomes '' == ''
[undefined] == false;     // true -- becomes '' == ''
[false] == false;         // false -- becomes 'false' == 0, then NaN == 0
```

Returning values

- All these methods can return the first element of a 2-element Array: `Array.prototype.shift`, `Array.prototype.splice`, `Array.prototype.slice`