W02 - Arrays, Logic, and Loops

- Array Literals
- Adding and removing values
- Array Methods
- Sets
- Maps
- If and else statements
- switch statements
- while and do...while Loops
- for loops
- Iterating over Collecte

Arrays - ordered list of values - built in object

- const myArray = [];
- const myArray = new Array();

Array Literals - create using [] with contents already

```
const fruit = ['banana','apple','orange'];
delete fruit[1]; // fruit now => ['banana','apple', undefined]
```

* deleting a value does not remove the space

Destructuring Arrays - assign multiple values at same time

```
Const [x, y] = [1,2]; // x == 1  y == 2
[x,y] = [y, x] ; //  y == 1  x == 2
```

Properties and Methods

.length ← Mutable, meaning can manually change it
making arrays shorter will remove extra values

.pop() ← removes and returns last element in array

.shift() ← removes and returns first element in array

.push() ← adds new value to end of array

.concat() ← Merge an array with one or more arrays, creating a new array

```
fruit = fruit.concat(['pineapple','kiwi']);
```

Spread operator – like concat

```
fruit = [...fruit...['pineapple','kiwi']];
```

.join() ← turns array into string containing all members of the array. 'banana, apple, orange, pineapple, kiwi'

- Use a separator other than comma .join('&');

.slice() ← creates a sub-array – Non-destructive
third and forth element = .slice(2,4)

.splice() ← first number - value where to start splice
Second number - how many values to remove
every value after is inserted

W02- Arrays, Logic, and Loops

.reverse() ← changes order of values permanently

.sort() ← alphabetically sorts values

```
[5, 9, 10].sort(); // [10, 5, 9]
```

.indexOf() ← return index of element or -1 if not in array

.includes() ← returns boolean

Multidimensional Array

```
const coordinates = [[1, 3], [4, 2]]
coordinates[0][0]; // returns 1 - first value of first array
```

use spread operator to flatten multidimensional arrays

SETS ← collection of unique values (no duplicates)

- currently, no literal notation for creating a set

```
const list = new Set(); // creates a set (empty)
```

.add(); ← adds value to set can add multiple

```
list.add(1).add(2).add(3);
```

adding a value already in set will ignore

```
const numbers = new Set([1, 2, 3]); // creates set with array
const letters = new Set('hello'); // letters == {'h', 'e', 'l', 'o'}
```

adding words to a set requires add() method

.size() ← return number of members in set

.has() ← return boolean ← very efficient

Sets do not have index notation

.delete() ← removes member of set

.clear() ← empties set

use spread operator to convert set to array

```
const arrayExample = [...setExample]
array.from(setExample);
const duplicate = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 9];
const nonDuplicate = [... new Set(duplicate)]; // 3, 1, 4, 5, 9, 2, 6
```

W02 - Arrays, Logic, and loops

Weak Sets - avoid problem with garbage collection but cannot have primitive datatypes

weakSet()

MAPS - Maps can use any datatype as a key

- must use .get() method to retieve values
- .set() method to add values to map

```
romanNumerals.set(1,'I'); // can repeatedly call set()
```

- .get() ← lookup value romanNumerals.get(4) returns 'IV'
- .has() ← return boolean

LOGIC - logical conditions

if Statements `if (condition) { }`

Ternary Operator `condition ? (what happens when true) : (false)`

- can be placed in a template string

switch

```
switch (number) {
    case 4:
        what to do
        break;
    case 5:
        what todo
        break;
    default
        what todo
        break;
}
```

Loops

```
while (condition) {    }
do {    } while (condion)
for (initialization; condition; after) {    }
    nesting for loop,
```

W02 - Arrays, Logic, and Loops

Looping over Arrays ← use Array indices

for (const value of arrayName) { do something }

Looping over Sets ← enumerable, sets can be iterated over each value in the set.

for-of loop: const letters = Set ('hello');

for (const letter of letters) {

do something using 'letter' as variable

} // will iterate through h,e,l,o

* weak sets aren't enumerable and cannot be iterated

Looping over maps; map have a keys() method.

for (const key of map.keys()) {

do something → iterate through keys }

for (const values of map.values()) {

do something → iterate through values }

for (const [key,value] of map.entries()) {

do something → iterate through key/value pairs }

* weak maps are non-enumerable