Sets

Level 4 – Section 3

Limitations With Arrays

Arrays don’t enforce uniqueness of items. Duplicate entries are allowed.

```
let tags = [];

tags.push( "JavaScript" );
tags.push( "Programming" );
tags.push( "Web" );
tags.push( "Web" );

console.log( "Total items ", tags.length );
```

Duplicate entry

```
> Total items 4
```

Using Set

ES2015

The *Set* object stores **unique** values of **any type**, whether primitive values or object references.

```
let tags = new Set();

tags.add("JavaScript");
tags.add("Programming");
tags.add({ version: "2015" });
tags.add("Web");
tags.add("Web");

console.log("Total items ", tags.size);
```

Both primitive values and objects are allowed

Duplicate entries are ignored

```
> Total items 4
```

We use the add() method to add elements to a Set

Using Set as Enumerable Object

Set objects are **iterable,** which means they can be used with *for...of* and destructuring.

```
let tags = new Set();

tags.add("JavaScript");
tags.add("Programming");
tags.add({ version: "2015" });
tags.add("Web");

for(let tag of tags){
  console.log(tag);
}
```

```
> JavaScript
> Programming
> { version: '2015' }
> Web
```

```
let [a,b,c,d] = tags;
console.log(a, b, c, d);
```

```
> JavaScript Programming { version: '2015' } Web
```

Effectively extracting elements via destructuring

WeakSet

The *WeakSet* is a type of *Set* where **only objects** are allowed to be stored.

```
let weakTags = new WeakSet();

weakTags.add("JavaScript");
weakTags.add({ name: "JavaScript" });
let iOS = { name: "iOS" };
weakTags.add(iOS);

weakTags.has(iOS);
weakTags.delete(iOS);
```

```
> TypeError: Invalid value used in weak set
```

Only objects can be added

```
> true
> true
```

WeakSets don't prevent the garbage collector from collecting entries that are no longer used in other parts of the system

Can’t Read From a WeakSet

WeakSets **cannot** be used with *for...of* and they offer **no** methods for reading values from it.

```
let weakTags = new WeakSet();

weakTags.add({ name: "JavaScript" });
let iOS = { name: "iOS" };
weakTags.add(iOS);

for(let wt of weakTags){
  console.log(wt);
}
```

```
> TypeError weakTags[Symbol.iterator]
  is not a function
```

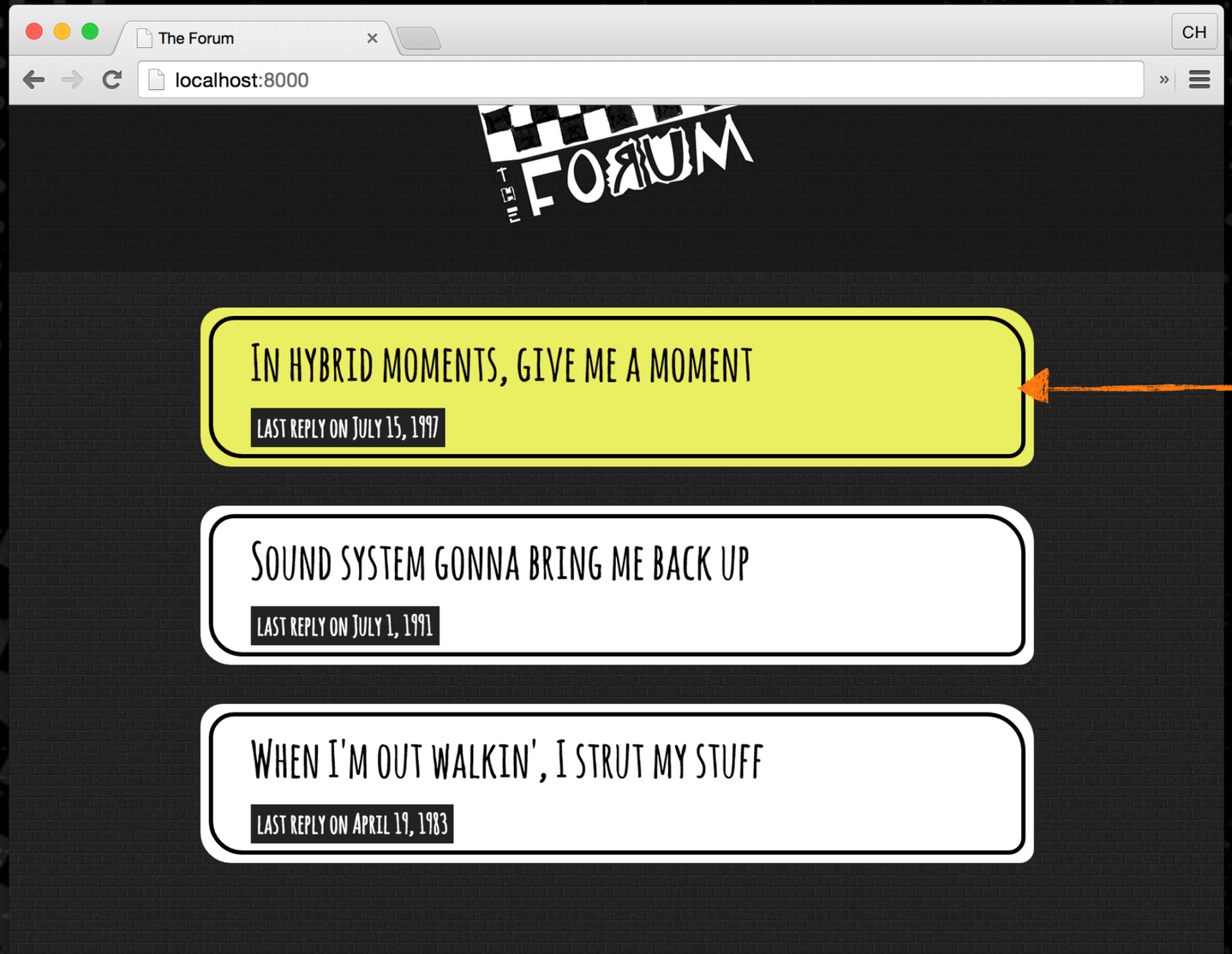
Not iterable!

If we can’t read values from a WeakSet,

when should we use them ?

Using WeakSets to Show Unread Posts

We want to add a different background color to posts that **have not yet been read.**



Unread posts should have a different background color

Using WeakSets to Show Unread Posts

One way to “tag” unread posts is to **change** a property on each post object once they are read.

```
let post = { //... };

//...when post is clicked on
postList.addEventListener('click', (event) => {
  //....
  post.isRead = true;
});

// ...rendering list of posts
for(let post of postArray){
  if(!post.isRead){
    _addNewPostClass(post.element);
  }
}
```

Mutates post object in order to indicate it’s been read

Checks a property on each post object

Showing Unread Posts With WeakSets

We can use *WeakSets* to create special groups from existing objects **without mutating them.** Favoring **immutable** objects allows for much **simpler** code with **no unexpected side effects.**

```
let readPosts = new WeakSet();

//...when post is clicked on
postList.addEventListener('click', (event) => {
  //....
  readPosts.add(post);
});

// ...rendering posts
for(let post of postArray){
  if(!readPosts.has(post)){
    _addNewPostClass(post.element);
  }
}
```

Adds object to a group of read posts

The has() method checks whether an object is present in the WeakSet