Arrays

Level 4 – Section 1

Assigning From Array to Local Variables

We typically access array elements by their index, but doing so for more than just a couple of elements can quickly turn into a **repetitive task.**

```
let users = ["Sam", "Tyler", "Brook"];

let a = users[0];
let b = users[1];
let c = users[2];

console.log( a, b, c );
```

```
> Sam Tyler Brook
```

This will keep getting longer as we need to extract more elements

Reading Values With Array Destructuring

We can use destructuring to assign **multiple values** from an array to local variables.

```
let users = ["Sam", "Tyler", "Brook"];
let [a, b, c] = users;
console.log( a, b, c );
```

```
> Sam Tyler Brook
```

Still easy to understand AND less code

Values can be **discarded**

```
let [a, , b] = users;
console.log( a, b );
```

```
> Sam Brook
```

Notice the blank space between the commas

Combining Destructuring With Rest Params

We can **combine** destructuring with rest parameters to **group values** into other arrays.

```
let users = ["Sam", "Tyler", "Brook"];
let [ first, ...rest ] = users;
console.log( first, rest );
```

```
> Sam ["Tyler", "Brook"]
```

Groups remaining arguments in an array

Destructuring Arrays From Return Values

When returning arrays from **functions**, we can assign to **multiple variables** at once.

```
function activeUsers(){
  let users = ["Sam", "Alex", "Brook"];
  return users;
}
```

Returns an array, as expected...

```
let active = activeUsers();
console.log( active );
```

```
> ["Sam", "Alex", "Brook"]
```

...or assigns to **individual variables**. Handy!

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

```
> Sam Alex Brook
```

Using for...of to Loop Over Arrays

The *for...of* statement iterates over **property values**, and it's a better way to loop over arrays and other **iterable objects.**

```
let names = ["Sam", "Tyler", "Brook"];
```

✗

```
for(let index in names){
  console.log( names[index] );
}
```

```
> Sam Tyler Brook
```

Uses index to read actual element

✓

```
for(let name of names){
  console.log( name );
}
```

```
> Sam Tyler Brook
```

Reads element directly and
with less code involved

Limitations of for...of and Objects

The *for...of* statement **cannot** be used to iterate over properties in plain JavaScript objects out-of-the-box.

```
let post = {
  title: "New Features in JS",
  replies: 19,
  lastReplyFrom: "Sam"
};

for(let property of post){
  console.log( "Value: ", property );
}
```

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

How do we know when it's okay to use *for...of* ***?***

Objects That Work With for...of

In order to work with *for...of*, objects need a special function assigned to the *Symbol.iterator* property. The presence of this property allows us to know whether an object is **iterable.**

```
let names = ["Sam", "Tyler", "Brook"];

console.log( typeof names[Symbol.iterator] );

for(let name of names){
  console.log( name );
}
```

Symbols are a new data type guaranteed to be unique

```
> function
```

Since there's a function assigned, then the names array will work just fine with for...of

```
> Sam
> Tyler
> Brook
```

Objects That Don't Work With for...of

No function assigned to the *Symbol.iterator* property means the object is **not iterable.**

```
let post = {
  title: "New Features in JS",
  replies: 19,
  lastReplyFrom: "Sam"
};

console.log( typeof post[Symbol.iterator] );

for(let property of post){
  console.log( property );
}
```

```
> undefined
```

Nothing assigned to Symbol.iterator, so the post object will not work with for...of

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

Finding an Element in an Array

Array.find returns the **first element** in the array that satisfies a provided testing function.

```
let users = [
  { login: "Sam",     admin: false },
  { login: "Brook",   admin: true  },
  { login: "Tyler",   admin: true  }
];

let admin = users.find( (user) => {
  return user.admin;
});

console.log( admin );
```

How can we find an admin in this array of users?

Returns first object for which user.admin is true

```
> { "login" : "Brook", "admin" : true }
```

One-liner arrow function

```
let admin = users.find( user => user.admin );
console.log( admin );
```

```
> { "login" : "Brook", "admin" : true }
```