LECTURE 12

Array Methods and Processing

Saying what you want done to a list, not how to loop through it — the highest-leverage skill for the rest of this course.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Iterating over arrays with forEach and for...of
2. Transforming with map, selecting with filter, aggregating with reduce
3. Searching and testing with find, some, and every, and ordering with sort
4. Frequently used string and object methods
5. Method chaining — expressing a data pipeline in one readable expression
6. Writing in an immutable, functional style, and avoiding common mutation mistakes

forEach

```
const fruits = ["apple", "banana", "cherry"];

fruits.forEach((fruit, index) => {
  console.log(`${index}: ${fruit}`);
});
// 0: apple
// 1: banana
// 2: cherry
```

forEach runs a function once per element. It always returns undefined -- use it purely for side effects, like logging.

for...of

```
for (const fruit of fruits) {
  if (fruit === "banana") continue;
  console.log(fruit);
}
```

for...of is a loop construct (not a method) that iterates over the VALUES of any iterable, and supports break/continue, unlike forEach.

for...in vs. for...of

NOTE

for...in iterates over an object's KEYS (or an array's indexes) and is generally used for plain objects. for...of iterates over VALUES and is generally preferred for arrays. Mixing them up is a common source of bugs.

map -- Same Length In, Same Length Out

```
const prices = [100, 200, 300];
const withTax = prices.map(price => price * 1.15);
console.log(withTax); // [115, 230, 345]
console.log(prices);  // [100, 200, 300] -- original untouched
```

map creates a NEW array by applying a function to every element -- ideal for converting every item into something else.

filter -- Keep Only What Matches

```
const numbers = [1, 2, 3, 4, 5, 6];
const evens = numbers.filter(n => n % 2 === 0);
console.log(evens); // [2, 4, 6]
```

filter creates a NEW array with only the elements for which the function returns true -- the result can be shorter than the original.

reduce -- Boil It Down to One Value

```
const cart = [10, 20, 30];
const total = cart.reduce((accumulator, price) => accumulator + price, 0);
console.log(total); // 60
```

reduce takes the combining function and an INITIAL VALUE for the accumulator -- here, 0.

Walking Through reduce, Step by Step

Step	accumulator	currentValue	returns
1	0	10	10
2	10	20	30
3	30	30	60

reduce Can Build Objects Too

```
const words = ["cat", "dog", "cat", "bird", "dog", "cat"];
const counts = words.reduce((acc, word) => {
  acc[word] = (acc[word] || 0) + 1;
  return acc;
}, {});
console.log(counts); // { cat: 3, dog: 2, bird: 1 }
```

reduce is more powerful than it looks -- you can use it to count occurrences, or even implement map/filter yourself.

find, some, and every

```
const users = [
  { id: 1, name: "Ali", active: true },
  { id: 2, name: "Sara", active: false },
  { id: 3, name: "Bilal", active: true },
];

users.find(u => u.id === 2);          // { id: 2, name: "Sara", ... }
users.some(u => u.active === false);  // true -- AT LEAST ONE matches
users.every(u => u.active === true);  // false -- not ALL match
```

find returns the FIRST matching element (or undefined); some/every return booleans.

sort -- Mutates in Place

```
const nums = [40, 1, 5, 200];
console.log(nums.sort());                  // [1, 200, 40, 5] -- WRONG (strings!)
console.log(nums.sort((a, b) => a - b)); // [1, 5, 40, 200] -- ascending
console.log(nums.sort((a, b) => b - a)); // [200, 40, 5, 1] -- descending
```

Without a comparison function, sort converts elements to strings first. (a,b)=>a-b is negative when a comes first, positive when b comes first.

sort Mutates the Array

WARNING

Unlike map and filter, sort (and also reverse, push, pop, splice) change the original array instead of returning a new one. If you need to keep the original order intact, sort a copy: [...nums].sort((a, b) => a - b).

Frequently Used String Methods

```
const text = "  Hello, World!  ";
text.trim();                // "Hello, World!"
text.toLowerCase();         // "  hello, world!  "
text.includes("World");     // true
text.trim().split(", ");    // ["Hello", "World!"]
"5".padStart(3, "0");       // "005"
`Hi ${"Ali"}`.slice(0, 2); // "Hi"
```

Frequently Used Object Methods

```
const person = { name: "Ayesha", age: 22 };
Object.keys(person);   // ["name", "age"]
Object.values(person); // ["Ayesha", 22]
Object.entries(person); // [["name","Ayesha"], ["age",22]]

const merged = Object.assign({}, person, { age: 23 });
console.log(merged); // { name: "Ayesha", age: 23 } -- person unchanged
```

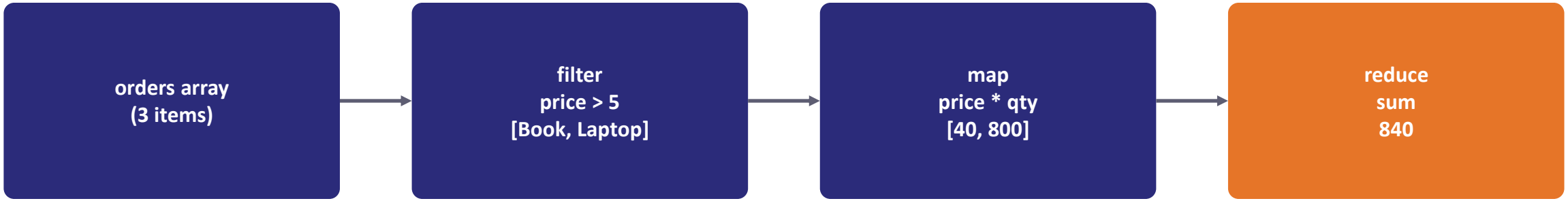
One Readable Multi-Step Pipeline

```
const orders = [
  { item: "Book", price: 20, qty: 2 },
  { item: "Pen", price: 2, qty: 10 },
  { item: "Laptop", price: 800, qty: 1 },
];

const totalForExpensiveItems = orders
  .filter(order => order.price > 5)              // keep Book, Laptop
  .map(order => order.price * order.qty)          // [40, 800]
  .reduce((sum, lineTotal) => sum + lineTotal, 0); // 840
```

Because map, filter, and similar methods return NEW arrays, you can call another array method directly on the result.

The Pipeline, Step by Step



Each stage takes the previous stage's output array as its input -- filter narrows the list, map transforms it, reduce collapses it to one number.

Read Chains Top to Bottom

TIP

When a chain gets long, put each method call on its own line. It reads like a numbered list of steps, and makes it much easier to see which step introduced a bug.

Mutating vs. Immutable Style

```
// Mutating (avoid)
function addItemMutating(cart, item) {
  cart.push(item); // changes the original array
  return cart;
}

// Immutable (prefer)
function addItemImmutable(cart, item) {
  return [...cart, item]; // returns a brand-new array
}

const cart = ["Book"];
const newCart = addItemImmutable(cart, "Pen");
console.log(cart);    // ["Book"] -- untouched
console.log(newCart); // ["Book", "Pen"]
```

The Same Idea for Objects

```
const settings = { theme: "light", fontSize: 14 };
const updated = { ...settings, theme: "dark" }; // new object, settings unchanged
```

map, filter, and reduce are all immutable by nature -- they never touch the original array. This is required by frameworks like React, which detect changes by comparing old and new data.

Methods That Mutate vs. Methods That Don't

Mutates the original array	Returns a new array, leaves original alone
push, pop, shift, unshift	map, filter, slice, concat
splice	reduce (usually)
sort, reverse	spread ([...arr])

Mixing these two columns up is one of the most common sources of bugs in JavaScript array code.

Three More Frequent Mistakes

```
// 1. forEach always returns undefined
const doubled = [1, 2, 3].forEach(n => n * 2); // WRONG -- use map

// 2. Missing initial value in reduce on an empty array
const empty = [];
// empty.reduce((a, b) => a + b); // TypeError!
const safe = empty.reduce((a, b) => a + b, 0); // 0 -- safe

// 3. == instead of === inside filter/find callbacks
const strNums = ["1", "2", "3"];
strNums.filter(n => n === 2); // [] -- "2" !== 2 (types differ)
```

KEY TAKEAWAYS

Lecture 12 in Seven Points

1. Use forEach or for...of to just walk through an array; use map, filter, and reduce to produce a new value.
2. map transforms every element (same length in/out); filter selects a subset; reduce combines it all into one result.
3. find returns the first match (or undefined); some/every return booleans about whether any/all elements match.
4. sort (like push, splice, and reverse) mutates the original array -- copy first with [...arr] to keep the original order.
5. Method chaining expresses multi-step data pipelines clearly, but put each call on its own line once a chain gets long.
6. Prefer an immutable, functional style -- return new arrays/objects instead of mutating existing ones, especially for React.
7. Always pass an initial value to reduce to avoid a runtime error on empty arrays.