Introduction to Angular Signals

Deborah Kurata
Developer

https://www.youtube.com/@deborah_kurata

```
let x = 5;
let y = 3;
let z = x + y;
console.log(z);
```

What is z?

8

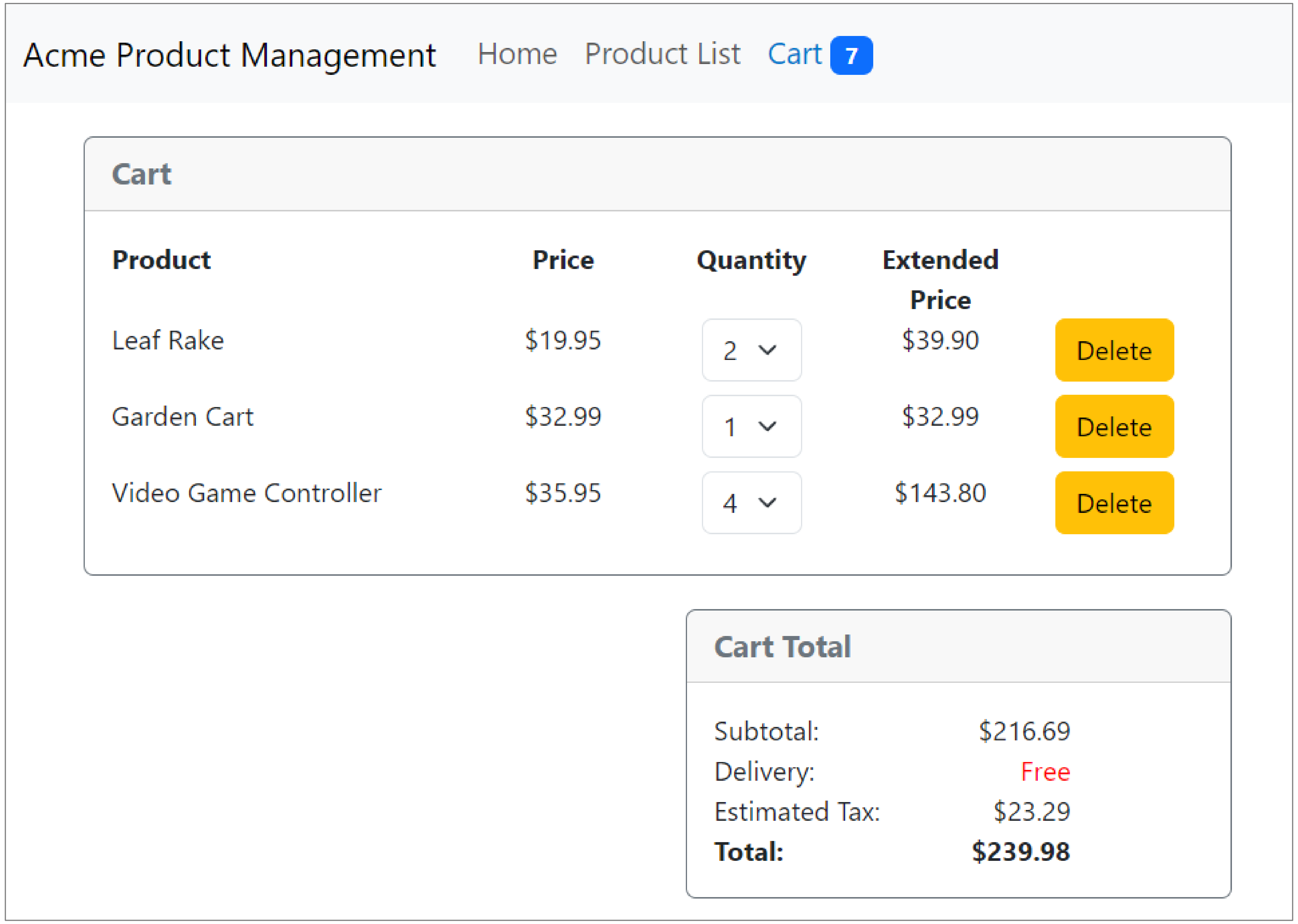
```
let x = 5;
let y = 3;
let z = x + y;
console.log(z);

x = 10;
console.log(z);
```

Now what is z?

8

Value is assigned when the expression is first evaluated

z does not react to changes in x or y



Acme Product Management Home Product List Cart 7

Cart

Product	Price	Quantity	Extended Price	
Leaf Rake	$19.95	2	$39.90	Delete
Garden Cart	$32.99	1	$32.99	Delete
Video Game Controller	$35.95	4	$143.80	Delete

Cart Total

Subtotal:	$216.69
Delivery:	Free
Estimated Tax:	$23.29
Total:	**$239.98**

We want to react to changes

Variables vs. Signals

Variables vs **Signals**

```
let x = 5;
let y = 3;
let z = x + y;

console.log(z); // 8

x = 10;
console.log(z); // 8
```

```
const x = signal(5);
const y = signal(3);
const z = computed(()=>
           x() + y());
console.log(z()); // 8

x.set(10);
console.log(z()); // 13
```

Signals

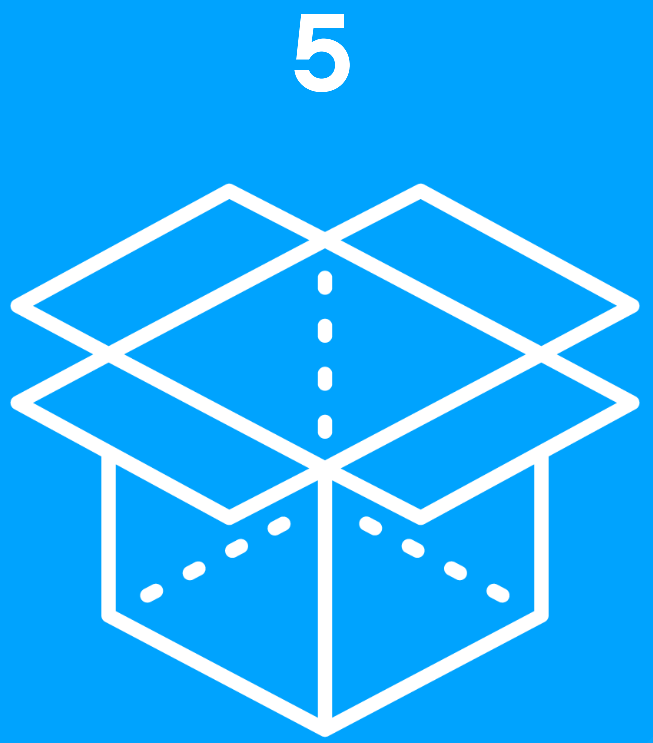
New way for our code to tell our templates (and other code) that our data changed

Make our code more reactive

Important for improved change detection

Available as a developer preview in Angular v16

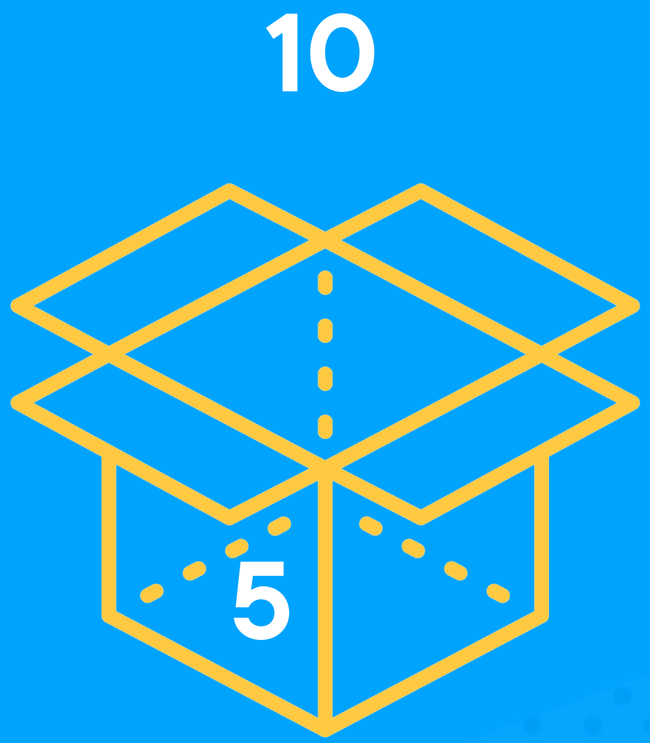
Signal = data value
+ change notification



Reactive Primitive

Signal = data value

+ change notification



Reactive Primitive

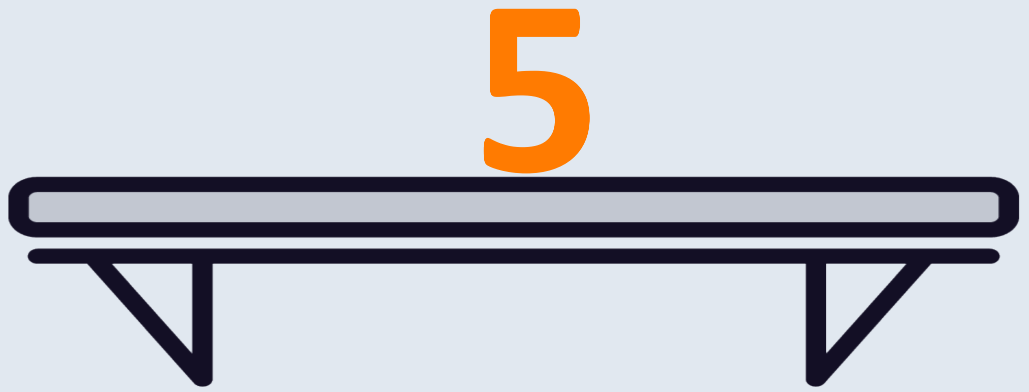
Overview

Examine the syntax for creating and reading a signal

Experiment with different ways to modify signals

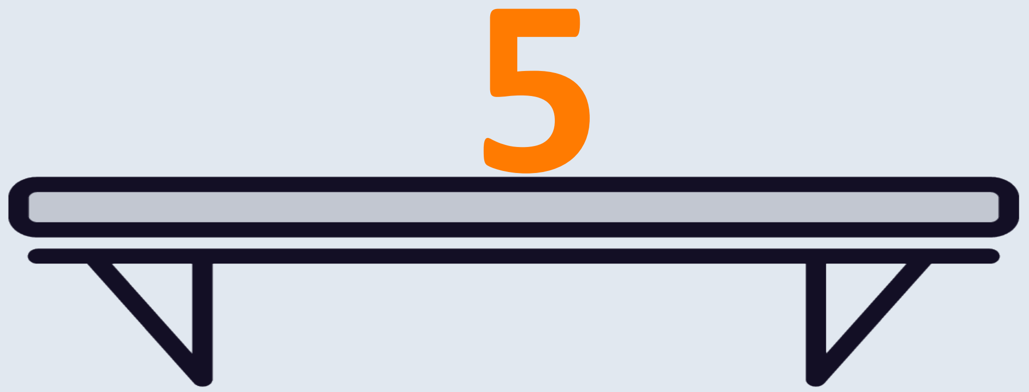
Explore how to build computed signals

Try out effects



```
let x = 5;
console.log(x);
```

```
let x = signal(5);
console.log(x());
```



```
let x = 5;
console.log(x);
```

```
let x = signal(5);
console.log(x());
```

Create a Signal

`signal`
constructor function

```
quantity = signal<number>(1);
```

Optional type:
string
number
array
object

Required initial value

Create a Signal

```
quantity = signal(1);
```

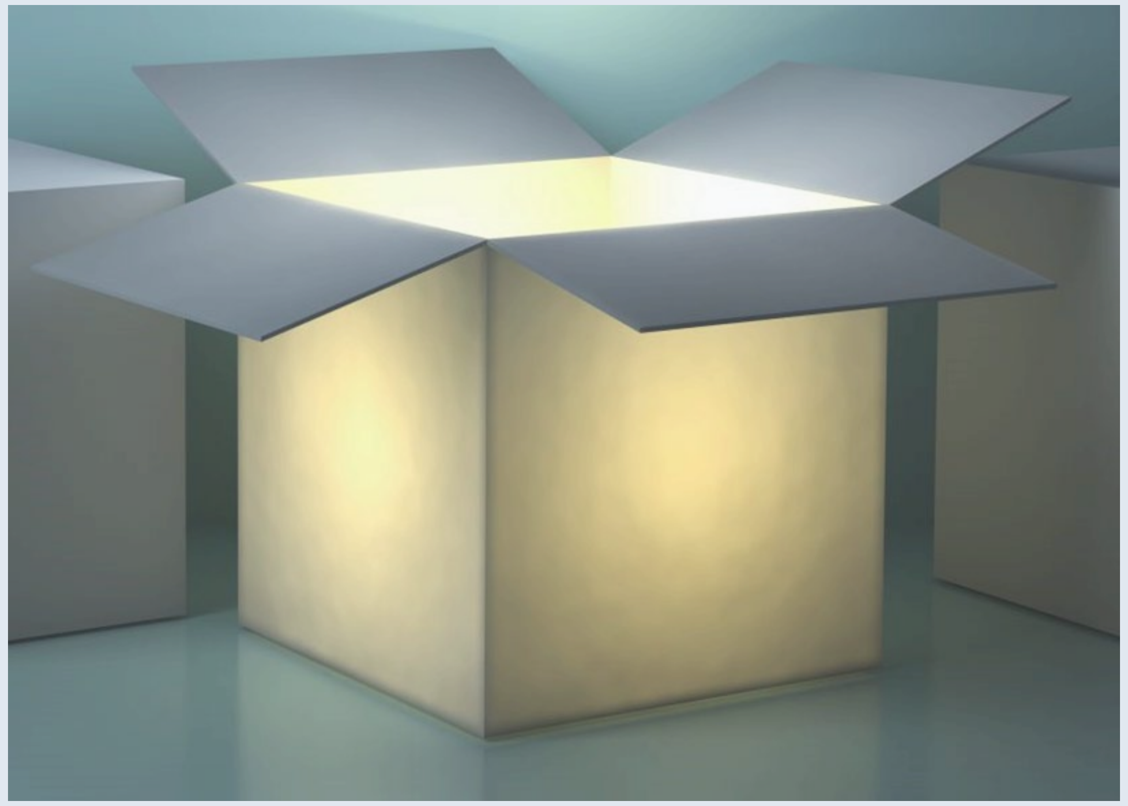
```
options = signal([1, 2, 3, 4, 5, 6]);
```

```
selectedProduct = signal<Product>({ id: 5,
                         name: 'Hammer', price: 8.9 });
```

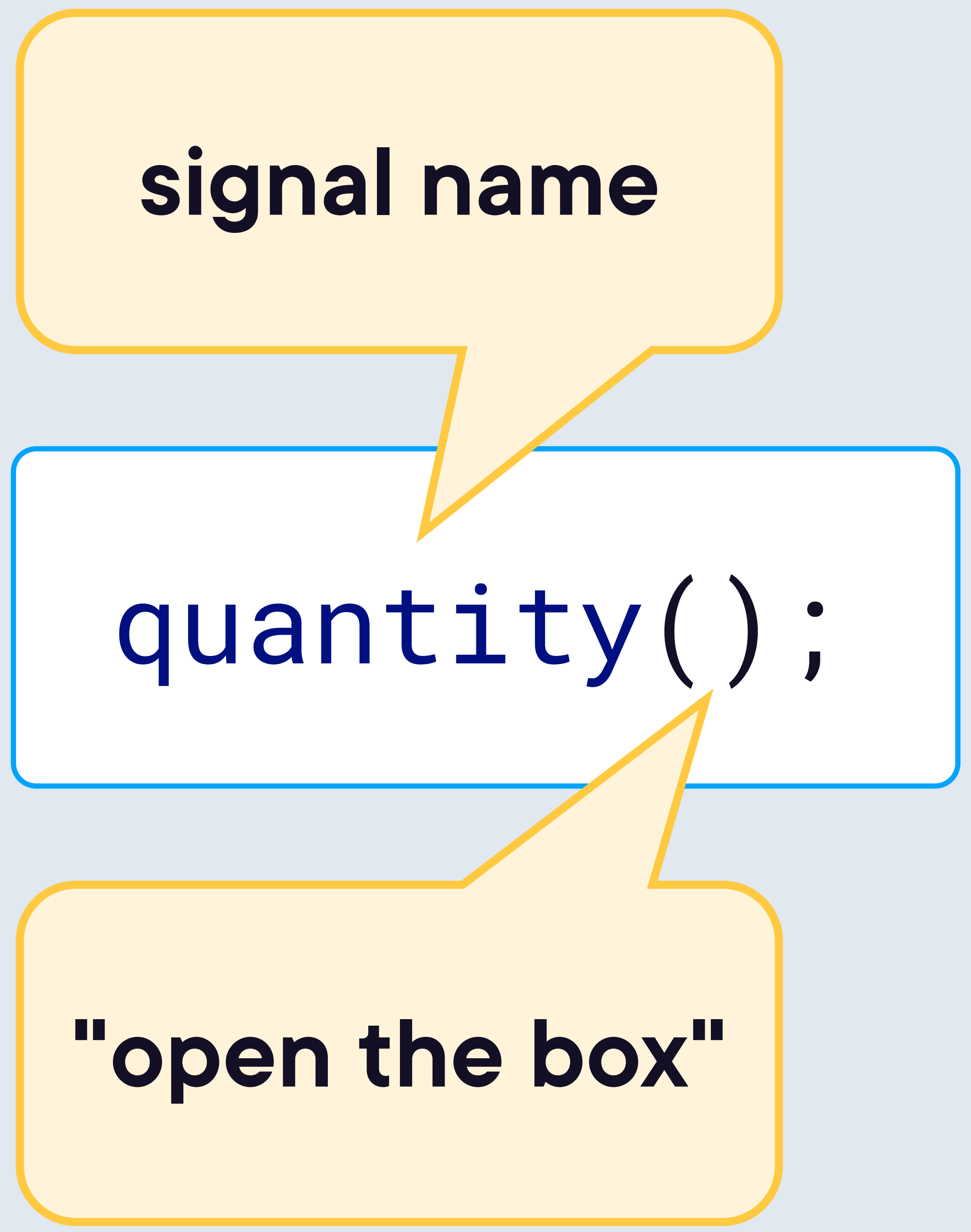
```
products = signal<Product[]>([]);
```

```
quantity = signal(1);
```

A signal created with the `signal()` constructor function is **writable**



Read a Signal



```
quantity();
```

Read a Signal

```
constructor() {
   console.log(this.quantity());
}
```

```
<option *ngFor="let opt of options()">
  {{ opt }}
</option>
```

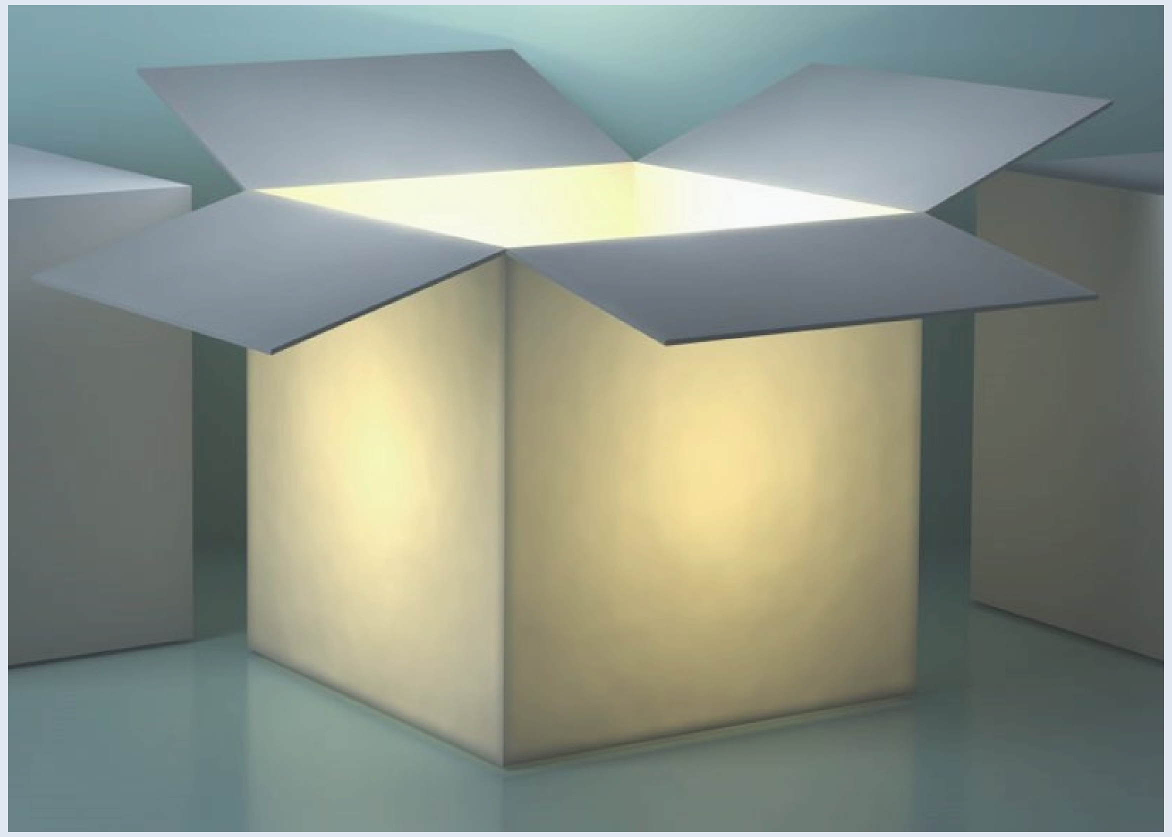
```
<div>Product: {{ selectedProduct().name}}</div>
<div>Price: {{ selectedProduct().price}}</div>
```

```
quantity();
```

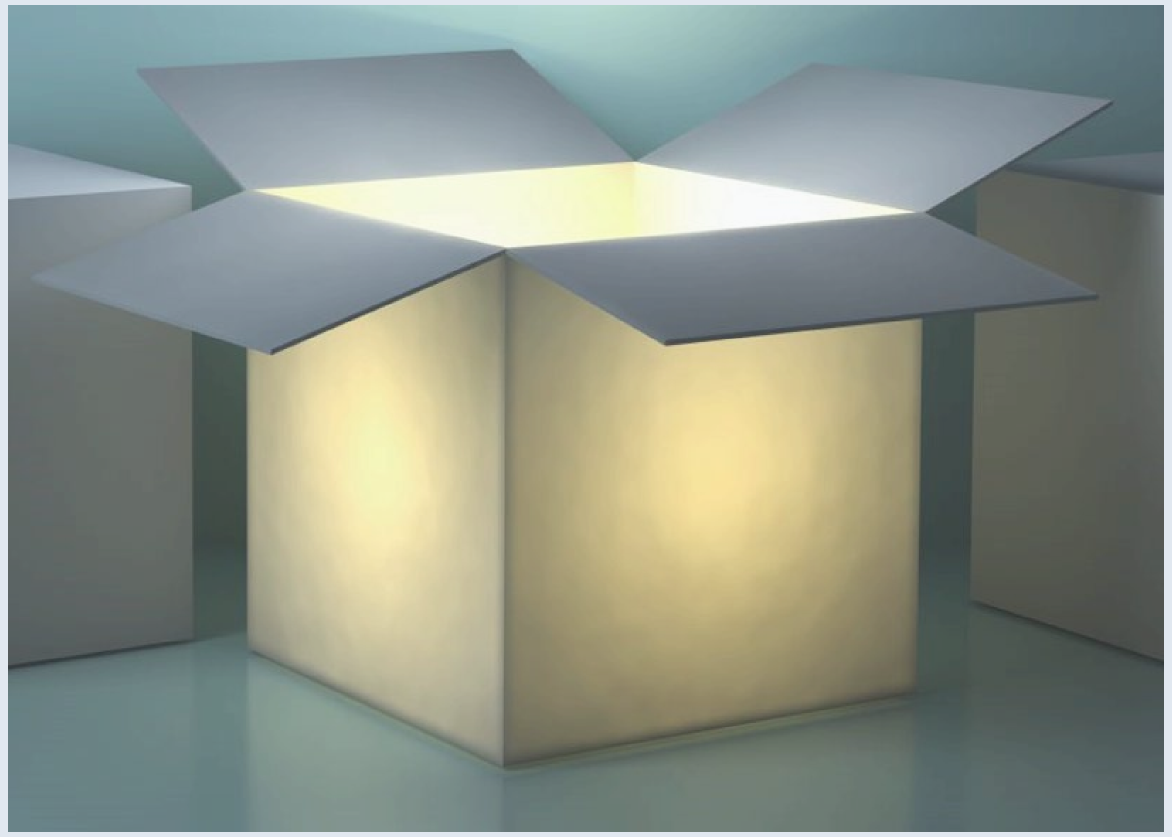
Reading a signal "opens the box" to get the current value

Reads the current value of the signal

Calls the signal's getter function



```
<div>Total: {{ exPrice() }}</div>
```



Reading a signal in a template:

- Returns the current signal value
- Registers the signal as a dependency of the template
- If the signal changes, the portion of the template is re-rendered

Demo

Create signals

Read signals

Modify a Signal

```
quantity = signal<number>(1);
```

```
// Replace the value
this.quantity.set(newQty);
```

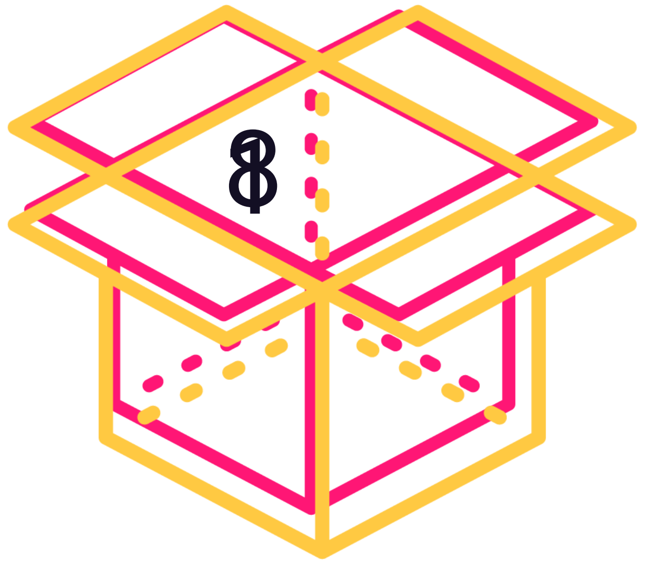
```
// Update value based on current value
this.quantity.update(qty => qty * 2);
```

Setting or Updating a Signal

Yeah, yeah!
I know you've changed
When I have a chance, I'll
get the current value

Hey!
I've changed the
signal value

```
{{ quantity() }}
```



```
quantity = signal(1);
```

```
this.quantity.set(8);
```

Template

Component

Reacting to Changes

```
onSomeEvent(qty: number) {
    this.quantity.set(qty);
    this.quantity.set(5);
    this.quantity.set(42);
}
```

Hey!
The quantity changed!

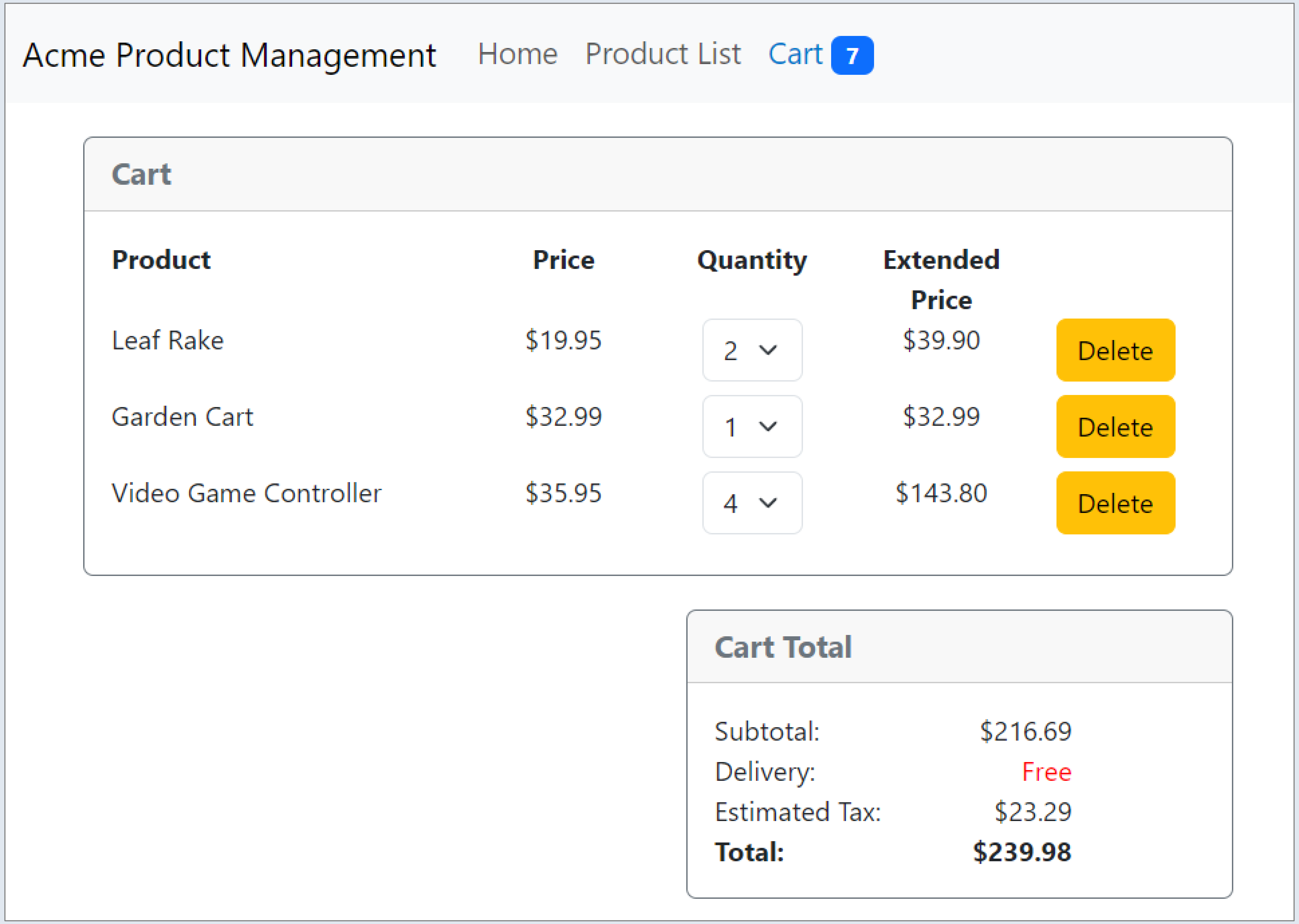
```
{{ quantity() }}
```

Only displays the **current value** when change detection is run (42)

Demo

Set and update signals

Computed Signals



Acme Product Management Home Product List Cart 7

Cart

Product	Price	Quantity	Extended Price	
Leaf Rake	$19.95	2	$39.90	Delete
Garden Cart	$32.99	1	$32.99	Delete
Video Game Controller	$35.95	4	$143.80	Delete

Cart Total

Subtotal:	$216.69
Delivery:	Free
Estimated Tax:	$23.29
Total:	**$239.98**

Define a Computed Signal

`computed` constructor function

Computation function

```
exPrice = computed(() =>
 this.selectedProduct().price * this.quantity());
```

Dependent signal

Dependent signal

Computed Signal

Creates a new signal that depends on other signals

Is read-only

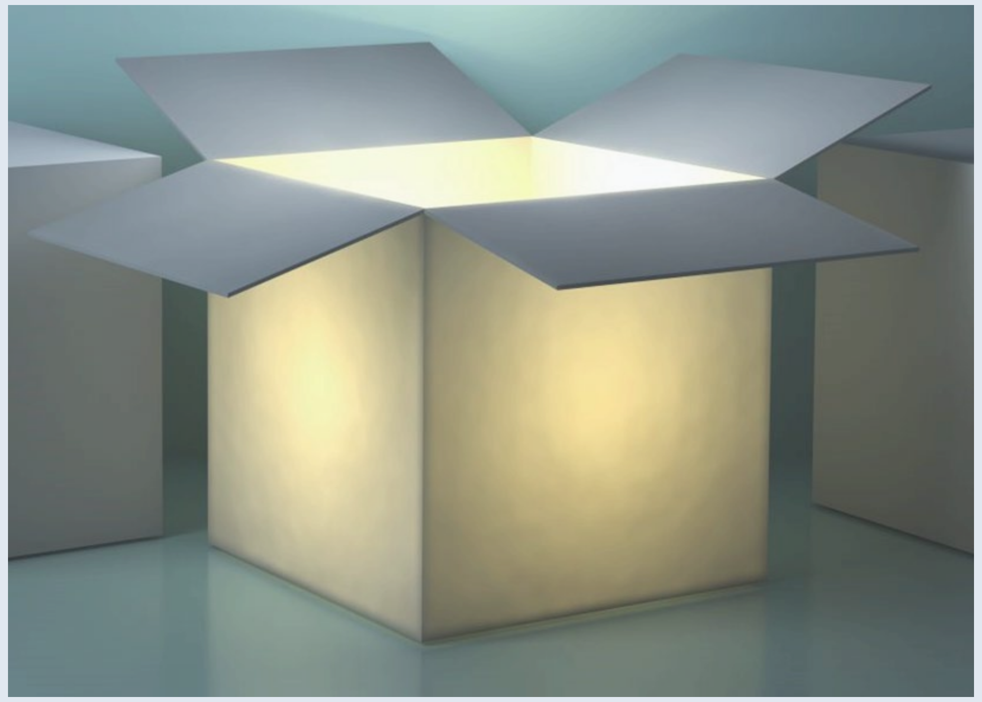
Recomputed if a dependent signal changes AND value is read

Should be side-effect free

```
<div *ngIf="exPrice()">
  Total: {{ exPrice() }}
</div>
```

Computed value is memoized, meaning it stores the computed result

That computed value is reused next time the computed signal is read



Demo

Define computed signals

Signal Effect

An effect is an operation that runs whenever one or more signal values change.

Defining an `effect`

`effect` **function**

Operation to execute

```
effect(() => console.log(this.selectedVehicle()));
```

Scheduled to re-run whenever any of the dependent signals change

Dependent signal

Defining an effect

```
effect(() => console.log(this.selectedVehicle()));
```

```
// Can be called declaratively
e = effect(() =>
            console.log(this.selectedProduct()));
```

Key Point

Use an effect to aid in debugging signals

```
effect(() => console.log(this.total()));
```

Signal Effect

Should not normally change state/value of a signal

Executes at least once
Tracks its dependent signals

When its dependencies change, effect is scheduled to be re-run

Will execute a minimum number of times
If an effect depends on multiple signals and several of them change, only one effect execution is scheduled

Demo

Try out some effects

What?

Signal = data value
+ change notification

How?

```
quantity = signal(1);
```

```
quantity();
```

```
// Replace the value
this.quantity.set(newQty);
```

```
// Update value based on current value
this.quantity.update(qty => qty * 2);
```

React to Changes

Use a **computed signal** to react and change state

```
exPrice = computed(() =>
        this.product().price * this.quantity());
```

Use an **effect** to react and execute code

```
effect(() => console.log(this.product()));
```

Read a signal in a **template** to react and re-render

```
{{ quantity() }}
```

Change Notifications

Register to receive change notifications by reading the signal

Computed signals and effects are scheduled to be re-run when their dependent signals change

Change detection is scheduled to re-render the view when any read signals change

When?

Use a **signal or computed signal** for any state (data) that could change

Put shared signals in **services**

Continue to use **observables** for async operations (`http.get`)

For More Information

Demo code

- https://stackblitz.com/edit/rxjs-signals-m10-deborahk

freeCodeCamp article

- https://www.freecodecamp.org/news/angular-signals

"Angular Signals: What? Why? and How?"

- https://youtu.be/oqYQG7QMdzw

Up Next:

Using Signals to Build a Shopping Cart Feature