

# Refactoring with Conditionals and Loops

---

(Rating, List of Products and Sorting)

# Outline and Learning Objectives

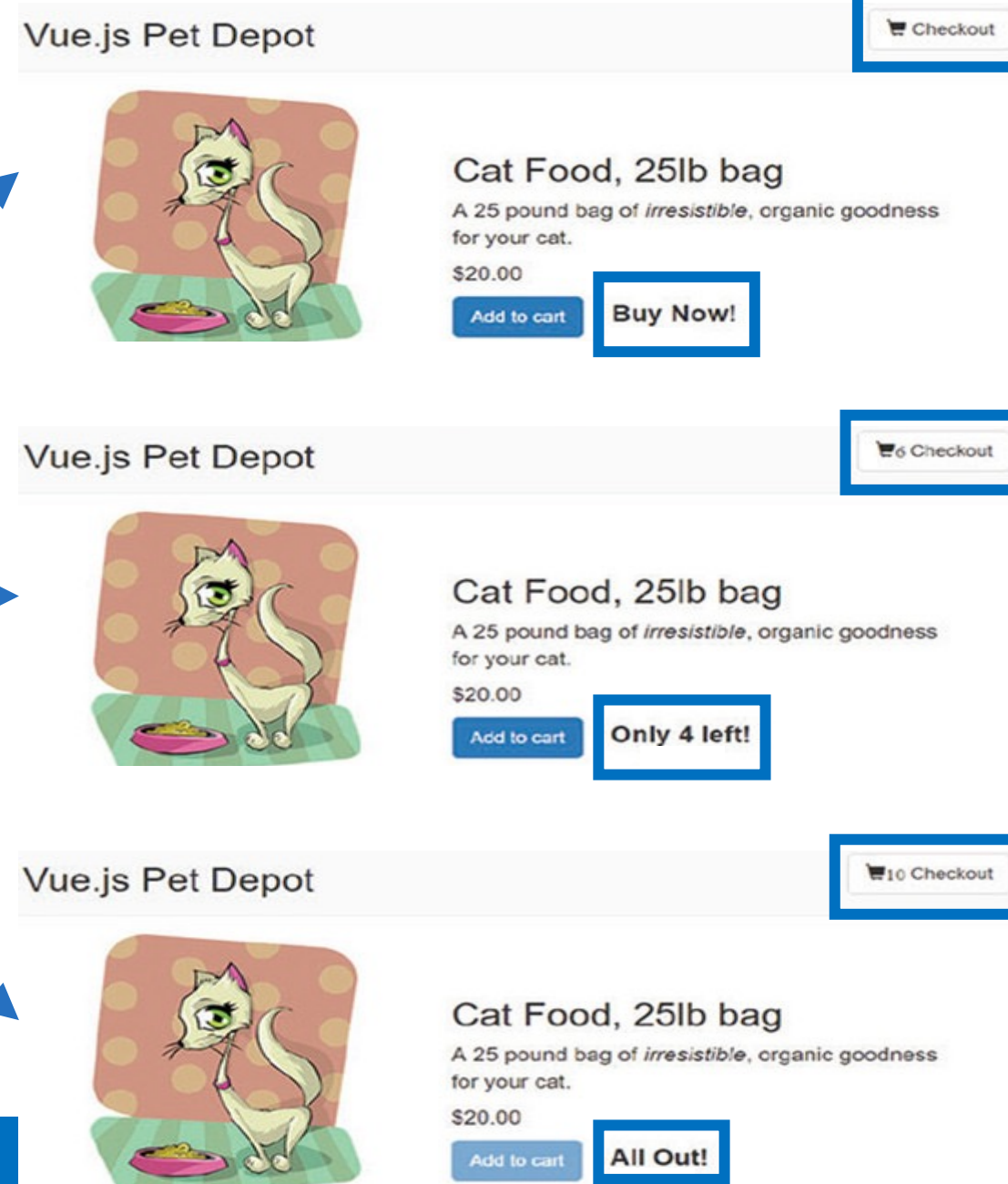
- **Conditionals:**
  - to master more advanced solutions with conditionals
  - **[Example]** managing different info displayed based on product availability
- **Managing List of Elements:**
  - to manage list of items in an application
  - to understand how to refactor applications for lists management
  - **[Example]** refactoring our app with a products list, and refactoring related functionalities
- **Sorting Elements:**
  - to master sorting of elements (ascending and descending orders)
  - **[Example]** sort our list of products
- **Suggestions for Reading**

# Conditionals

# Our Expected Result (Using Conditionals)

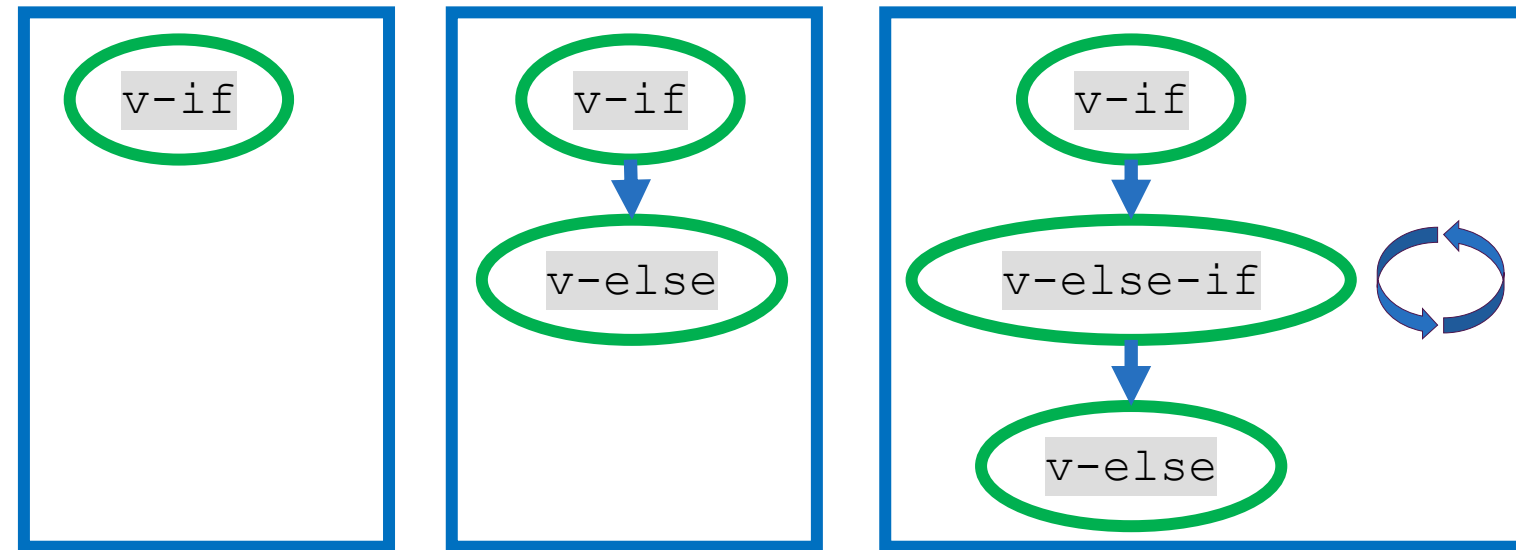
- **Our previous version:** no message
- **New version:** a different message for each case
- Availability: **10 products**
- **3 Cases:**
  - 1<sup>st</sup> Case: all items available
  - 2<sup>nd</sup> Case: less than 5 items available
  - 3<sup>rd</sup> Case: no items available

• **Solution:** we use **Conditionals**



# Conditionals: `v-if`, `v-else-if`, `v-else`

- `v-if` can be used alone
- If you use `v-else` (or `v-else-if`), it must immediately follow a `v-if` or `v-else-if`
- `v-else-if` can be more than once in a block



- **You cannot add extra elements in between, otherwise conditionals will not be recognized!**
- **Keep the logic in the view at the minimum**; instead, if you can use **computed properties** and **methods**, it is better:
  - it will make **your code easier to read**, more **understandable**, and easier to **maintain**

```
// This won't work
<p v-if="showMe">The if text</p>
<p>Other if text</p>
<p v-else>The else text</p>
```



```
// Nor will this
<div>
  <p v-if="showMe">The if text</p>
</div>
<div>
  <p v-else>The else text</p>
</div>
```



```
// Instead, consider grouping
<div v-if="showMe">
  <p>The if text</p>
  <p>Other if text</p>
</div>
<div v-else>
  <p>The else text</p>
</div>
```



# First Step: Increasing the Product Availability

- First, let's increase the inventory from 5 to 10:

```
product: {  
  id: 1001,  
  title: "Cat Food, 25lb bag",  
  description: "A 25 pound bag of ...",  
  price: 2000,  
  image: "assets/images/product-fullsize.png",  
  availableInventory: 10 // was '5'  
},
```

## 2<sup>nd</sup> Case: less than 5 items available

- When inventory is **less than 5**, we display a message showing how many are left.

- [First Solution]** we use an **expression** in the `v-if` directive

```
<button v-on:click='addToCart' v-if='canAddToCart'>
  Add to cart
</button>
<span v-if="product.availableInventory - cartItemCount < 5">
  Only {{product.availableInventory - cartItemCount}} left!
</span>
```

- Keep the logic in the view at the minimum**; instead, if you can use **computed properties** and **methods**, it is better
- [Better Solution]** it is also possible to use a computed property:

```
<span v-if="itemsLeft < 5">Only {{itemsLeft}} left!</span>
...
computed: {
  ...
  itemsLeft() {
    return this.product.availableInventory - this.cartItemCount;
  }
}
```

### Vue.js Pet Depot

7  Checkout



#### Cat Food, 25lb bag


A 25 pound bag of *irresistible*, organic goodness

Price: 2000

Available stock: 10

Add to cart Only 3 left!

# Further Improvements: the Other 2 Cases

10  Checkout




## Cat Food, 25lb bag

A 25 pound bag of irresistible, organic goodness

Price: 2000

Add to the Cart **Only 0 left!**

- [3<sup>rd</sup> Case: no items available] **Partially Solved**
  - **[Problem]** When the inventory reaches 0, the message displays “Only 0 left!”:
    - **not very clear message for the User**
    - **we need to improve the message**
- [1<sup>st</sup> Case: all items available] **to be solved:**
  - **currently no message**
  - **we need to add a message**, potentially one that **encourages the user to buy now**


 Checkout



## Cat Food, 25lb bag

A 25 pound bag of irresistible, organic goodness

Price: 2000

Add to the Cart 



# Solution: `v-if` , `v-else-if` , `v-else`

- All cases solved with conditionals and a computed property:

```
<span v-if='itemsLeft === 0'>All out!</span>  
<span v-else-if="itemsLeft < 5">Only {{itemsLeft}} left!</span>  
<span v-else>Buy now!</span>
```

🛒 Checkout



## Cat Food, 25lb bag

A 25 pound bag of irresistible, organic goodness

Price: 2000

Add to the Cart Buy now!

6 🛒 Checkout



## Cat Food, 25lb bag

A 25 pound bag of irresistible, organic goodness

Price: 2000

Add to the Cart Only 4 left!

10 🛒 Checkout



## Cat Food, 25lb bag

A 25 pound bag of irresistible, organic goodness

Price: 2000

Add to the Cart All out!

- `v-if`: when no more left, show 'All out'
- `v-else-if`: when inventory  $< 5$ , show how many left
- `v-else`: when inventory  $\geq 5$ , show 'Buy now'

- An alternative is to use a computed property to decide what message to display.

# Managing Lists of Elements (Part 1)

1. Display Rating of Products
2. List of Products

# Listing Elements with v-for

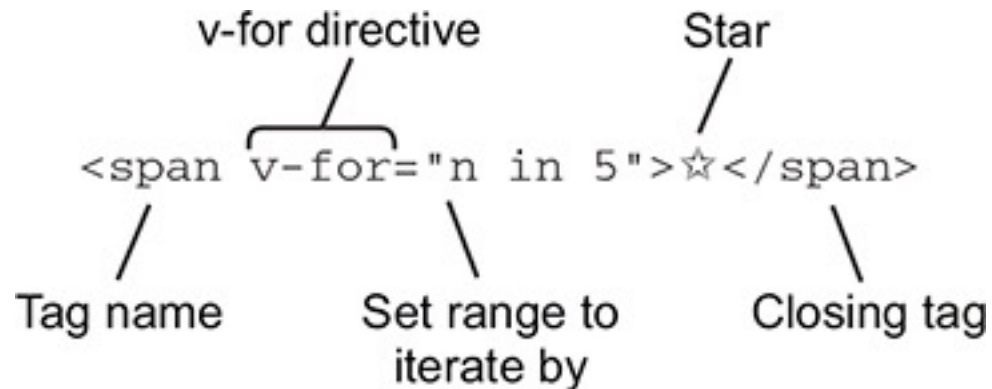
- We use **v-for** to:

1. add a **5-star rating system**
2. **list all the products** from an array

1. Adding star rating with **v-for range**:

- one way to use the **v-for** is by **giving it an integer**;
- When added to an element, it will be repeated as many times as the integer indicates;
- this is known as **v-for range**;

Add a **5-star rating system** with **v-for range**:



0 Checkout



## Cat Food, 25lb bag

A 25 pound bag of *irresistible*, organic goodness

Price: 2000

Available stock: 10



# A Rating Star System v-for Range

**[First Step]** add rating to product information

```
data: {  
  ...  
  product: {  
    ...  
    rating: 3  
  },  
}
```

**Use filled stars to represent rating:**

- **[Example]** a rating of 3 will have 3 filled stars and 2 empty ones;
- we achieve this by first repeat the **filled star** the 'rating' number of times;
- then **empty star** `5 - rating` number of times

```
<div>  
  <span v-for='n in product.rating'>★</span>  
  <span v-for='n in 5 - product.rating'>☆</span>  
</div>
```

0  Checkout



## Cat Food, 25lb bag

A 25 pound bag of *irresistible*, organic goodness

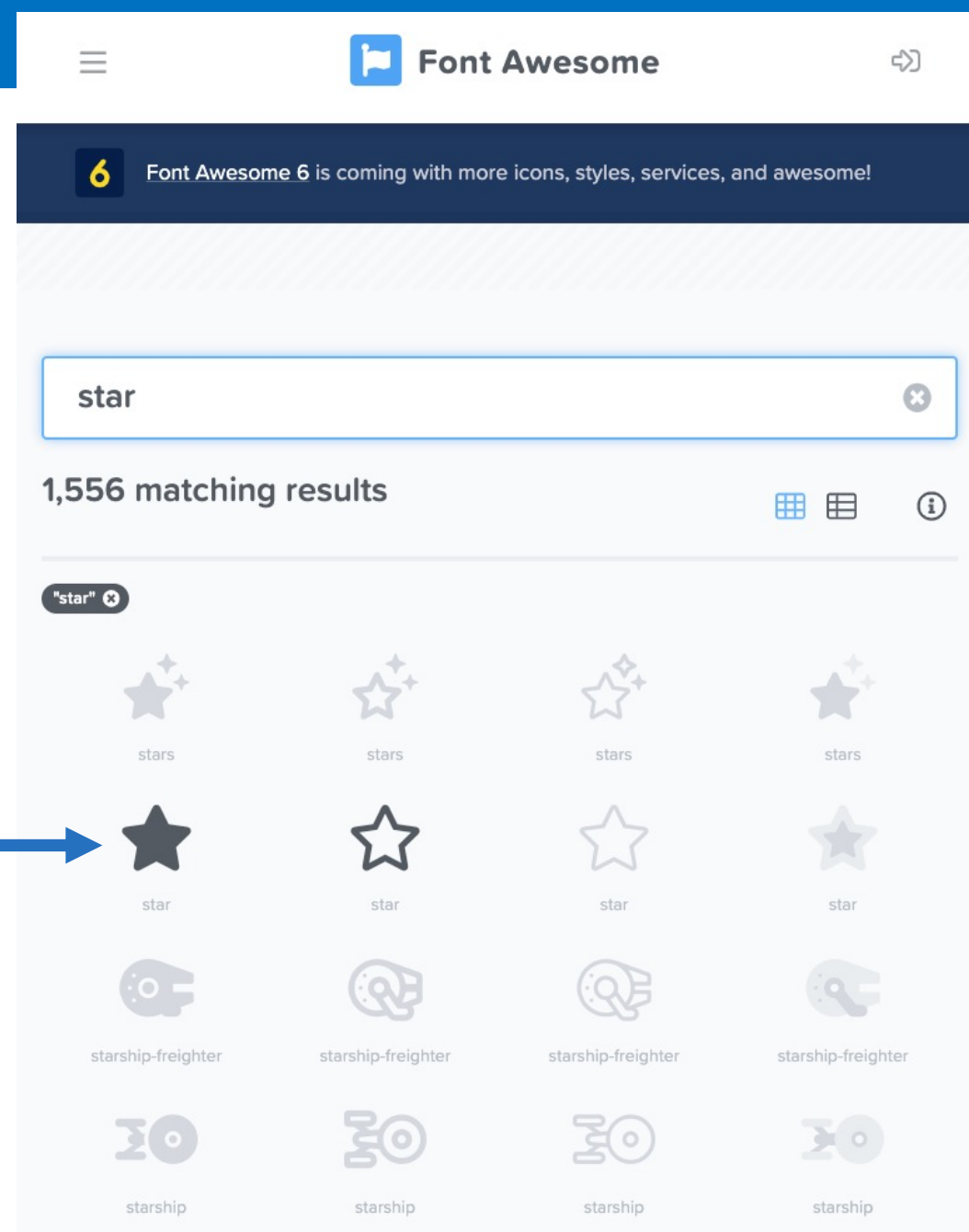
Price: 2000

Available stock: 10

Add to cart Buy now!



- There are **many ways to create a rating system**, for instance:
  - **[Alternative 1] [CSS (Textbook)]**  
the example in the **textbook** uses CSS class to achieve similar effect. **Filled stars and empty stars have different CSS classes**
  - **[Alternative 2] [Font Awesome]**  
the star symbol can also be achieved with the Font Awesome icon we discussed before.
  - **[Alternative 3] [Only Filled Stars]**  
you can just show the filled stars.



# Managing Lists of Elements (Part 2)

-  1. Display Rating of Products
- 2. List of Products



## Cat Food, 25lb bag



A 25 pound bag of *irresistible*, organic goodness for your cat.

Price: 2000

Available stock: 10

★★☆☆☆

## Yarn



Yarn your cat can play with for a very **long** time!

Price: 299

Available stock: 7

★★★☆☆

## Refactoring:

- the process that allows **restructuring code**, by keeping the same **external behavior**, **changing its architecture**, but **preserving its functionality** for:
  - **improving readability** and **maintainability** (for yourself, for your team)
  - **reducing complexity**

Our Objective (List of Products as Final Result)

## Refactoring Activities

1. Add products
2. Load products
3. Refactoring:
  1. show products with v-for (**most of the functions will stop working**)
  2. update **data** from **product** to **products**
  3. update “Add to Cart” button
  4. update canAddToCart **computed property** as a **method**
  5. add and use a **new cartCount method** for the different products
  6. **update all the methods using cartCount**
  7. Fixing the inventory message

# Adding and Loading More Products

**[1<sup>st</sup> Step: Add Products]** to add more products

- the `products` array is stored in a new JavaScript file `products.js`

```
let products = [  
  {  
    "id": 1001,  
    "title": "Cat Food, 25lb bag",  
    "description": "A 25 pound ...",  
    "price": 20.00,  
    ...  
    "availableInventory": 10,  
    "rating": 2  
  },  
  { "id": 1002, ... },  
  { "id": 1003, ... },  
  { "id": 1004, ... },  
  ...  
]
```

**[2<sup>nd</sup> Step: Load Products as a Script]** `products.js` needs to be loaded into our main html file

- just as any other JavaScript file or library
- `<script src="products.js"></script>`

**[3<sup>rd</sup> Step: Load Products as a Vue.js Property]** then the `products` array can be used in our Vue instance

- or anywhere in our html or javascript code


```
<script src="products.js"></script>
```

```
...  
data: {  
  ...  
  // The first 'products' is a Vue property  
  // The second products is the array from the 'products.js'  
  products: products,  
  ...  
}
```




# Show Products with v-for


```
<div v-if='showProduct'>
  <div v-for="product in products">
    <!-- product information -->
    <h2 v-text="product.title"></h2>
    ...
```




```
methods: {
  addToCart: function() {
    this.cart.push( this.product.id );
  }
}
```




```
<!-- 'add to cart' button -->
<button v-on:click='addToCart' v-if='canAddToCart'>Add to cart</button>
...
```




```
<!-- inventory message -->
<span v-if='product.availableInventory === cartItemCount'>All out!</span>
...
```



```
<!-- rating -->
<div><span v-for='n in product.rating'>★</span></div>
</div>
</div>
```



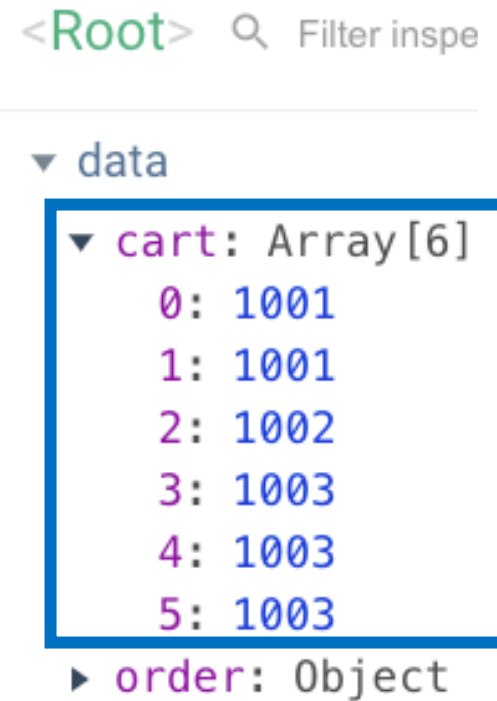
```
computed: {
  ...
  canAddToCart: function() {
    return this.product.availableInventory >
           this.cartItemCount;
  }
}
```



- When we apply this, **most of the important functionalities will not work anymore**
- This is because **functionalities were designed to work for 1 product**, now we have **n products**:
  - we need **to refactor different elements** to make them again to work, and **to have again the same behaviour expected before, for each of the products**

# Refactoring "Add to Cart" Functionality

- We need to update the `addToCart` method, so it takes the `product` in input and push its ID to the cart
  - `addToCart(product) { this.cart.push(product.id); }`
- And the `button` in our html page now looks like:
  - `<button v-on:click='addToCart(product)' ...>Add to cart</button>`
- We also need to update the `canAddToCart` computed property:
  - in this way, it checks the amount of each product in the cart;
  - first, we need to change it to a method, so it can take parameters
    - (computed properties cannot have input parameters)



```
methods: {  
  ...  
  canAddToCart(product) {  
    return product.availableInventory > this.cartCount(product.id);  
  },  
  ...  
}
```

- The `cartCount` is a new method that return the number of a product type in the cart

# The cartCount Method

- The `cartCount` is a new method that return the number of a product type in the cart

```
cartCount(id) {  
  let count = 0;  
  for(let i = 0; i < this.cart.length; i++) {  
    if (this.cart[i] === id) {  
      count++;  
    }  
  }  
  return count;  
}
```

- Now we can update the 'Add to cart' button to

```
<button v-on:click='addToCart(product)' v-if  
='canAddToCart(product)'> Add to cart</button>
```

<Root> 🔍 Filter inspe

▼ data

▼ cart: Array[6]

0: 1001

1: 1001

2: 1002

3: 1003

4: 1003

5: 1003

► order: Object

# Fixing the Inventory Message

- **The problem is that it still uses** the **total cart item count** to determine which messages to display.
- We need to change this code so that we now **calculate the message based on the cart item count of only that item.**
- To fix this issue, let's change from the `cartItemCount` computed property to our new `cartCount` method

```
...  
<div v-for="product in products">  
...  
  <span v-if='product.availableInventory === cartCount(product.id) '>All out!</span>  
  <span v-else-if="product.availableInventory - cartCount(product.id) < 5">  
    Only {{product.availableInventory - cartCount(product.id)}} left!  
  </span>  
  <span v-else>Buy now!</span>  
...
```

# Sorting Elements

# Sorting the Products by Price

- We need to create a **computed property** `sortedProducts` to return our **sorted results**
- The `v-for` then uses `sortedProducts` to display the product list
  - `<div v-for="product in sortedProducts">`
- As an example, we **sort by price from low to high (ascending order)** →
- This uses the **JavaScript array sort method**.

```
computed: {  
  sortedProducts() {  
  
    // the comparison function that defines the order  
    function compare(a, b) {  
      if (a.price > b.price) return 1;  
      if (a.price < b.price) return -1;  
      return 0;  
    }  
  
    // sort the 'products' array and return it  
    return this.products.sort(compare);  
  }  
}
```

- To have **descending**, same function, but `return 1` and `return -1` **inverted**
- Example of **ordering with text (alphabetical order)** is in our **textbook**

0  Checkout

## Yarn



Yarn your cat can play with for a very **long** time!

Price: 2.99

Available stock: 7

★★★★☆☆

## Cat House



A place for your cat to play!

Price: 7.99

Available stock: 11

# Suggestions for Reading

# Reading

Chapter 5 of the “**Vue.js in Action**” textbook



# Questions?