What is an Array of Objects?

Let me explain using your products example:

Understanding the Structure

An **Array of Objects** is a collection (list) where each item is an object containing related information.

javascript

```javascript
// This is an ARRAY (indicated by square brackets [])
const products = [
    // Each item inside is an OBJECT (indicated by curly braces
{})
    {
        id: 1,
        name: "PHANTOM X",
        category: "headphones",
        price: 349
    },
    {
        id: 2,
        name: "WAVE PRO",
        category: "speakers",
        price: 599
    }
    // ... more objects
];
```

Breaking it Down

1. The Array Part [ ]

- The outer square brackets [ ] create an array
- An array is like a numbered list

- Each position has an index (0, 1, 2, 3...)

javascript

```javascript
products[0]  // First product (PHANTOM X)
products[1]  // Second product (WAVE PRO)
products[2]  // Third product (PULSE MINI)
```

2. The Object Part `{ }`

- Each item in the array is an object with curly braces `{}`
- Objects store data as **key-value pairs**
- Keys are like labels, values are the actual data

javascript

```javascript
{
    id: 1,              // key: id,    value: 1
    name: "PHANTOM X",  // key: name,  value: "PHANTOM X"
    price: 349          // key: price, value: 349
}
```

Accessing Data

Access an entire object:

javascript

```javascript
products[0]
// Returns: {id: 1, name: "PHANTOM X", category: "headphones",
price: 349, ...}
```

Access a specific property:

javascript

```JavaScript
products[0].name      // "PHANTOM X"
products[0].price     // 349
products[1].category  // "speakers"
products[2].icon      // "🎵"
```

Why Use Array of Objects?

Instead of this (separate arrays):

javascript

```JavaScript
const names = ["PHANTOM X", "WAVE PRO", "PULSE MINI"];
const prices = [349, 599, 179];
const categories = ["headphones", "speakers", "earbuds"];
// Hard to keep track! What if they get out of sync?
```

We use this (array of objects):

javascript

```JavaScript
const products = [
    {name: "PHANTOM X", price: 349, category: "headphones"},
    {name: "WAVE PRO", price: 599, category: "speakers"},
    {name: "PULSE MINI", price: 179, category: "earbuds"}
];
// All related data stays together!
```

Real-World Analogy

Think of your products array like a **filing cabinet**:

```
Filing Cabinet (Array)
├── Folder #0 (Object)
```

```
|    ├── ID: 1
|    ├── Name: PHANTOM X
|    ├── Category: headphones
|    └── Price: $349
|
├── Folder #1 (Object)
|    ├── ID: 2
|    ├── Name: WAVE PRO
|    ├── Category: speakers
|    └── Price: $599
|
└── Folder #2 (Object)
     ├── ID: 3
     ├── Name: PULSE MINI
     ├── Category: earbuds
     └── Price: $179
```

Common Operations

Loop through all products:

javascript

```javascript
for (let i = 0; i < products.length; i++) {
    console.log(products[i].name);  // Prints each product name
}
```

Using map():

javascript

```javascript
products.map(product => {
    console.log(product.name);  // Prints each product name
});
```

Find a specific product:

javascript

```javascript
const headphones = products.filter(product => product.category
=== "headphones");
// Returns all headphone products
```

In Your Example

You have **12 objects** (products) inside **1 array**:

- Each object represents one product
- Each product has 7 properties: `id`, `name`, `category`, `price`, `description`, `icon`, `badge`
- You can easily loop through all 12 products
- You can access any product's information using its index and property name

This structure makes it easy to:

- Display all products on a webpage
- Search for specific products
- Filter by category
- Sort by price
- Add products to a shopping cart