The app in the previous example works, but there's a subtle bug — the store is subscribed to, but never unsubscribed. If the component was instantiated and destroyed many times, this would result in a *memory leak*.

Start by declaring `unsubscribe` in `App.svelte`:

```
const unsubscribe = count.subscribe(value => {
    countValue = value;
});
```

> *Calling a `subscribe` method returns an `unsubscribe` function.*

You now declared `unsubscribe`, but it still needs to be called, for example through the `onDestroy` lifecycle hook:

```
<script>
    import { onDestroy } from 'svelte';
    import { count } from './stores.js';
    import Incrementer from './Incrementer.svelte';
    import Decrementer from './Decrementer.svelte';
    import Resetter from './Resetter.svelte';

    let countValue;

    const unsubscribe = count.subscribe(value => {
        countValue = value;
    });

    onDestroy(unsubscribe);
</script>

<h1>The count is {countValue}</h1>
```

It starts to get a bit boilerplatey though, especially if your component subscribes to multiple stores. Instead, Svelte has a trick up its sleeve — you can reference a store value by prefixing the store name with `$`:

```
<script>
    import { count } from './stores.js';
    import Incrementer from './Incrementer.svelte';
    import Decrementer from './Decrementer.svelte';
    import Resetter from './Resetter.svelte';
</script>

<h1>The count is {$count}</h1>
```

> *Auto-subscription only works with store variables that are declared (or imported) at the top-level scope of a component.*

You're not limited to using `$count` inside the markup, either — you can use it anywhere in the `<script>` as well, such as in event handlers or reactive declarations.

> *Any name beginning with `$` is assumed to refer to a store value. It's effectively a reserved character — Svelte will prevent you from declaring your own variables with a `$` prefix.*