To run code when your component is destroyed, use `onDestroy`.

For example, we can add a `setInterval` function when our component initialises, and clean it up when it's no longer relevant. Doing so prevents memory leaks.

```
<script>
	import { onDestroy } from 'svelte';

	let counter = 0;
	const interval = setInterval(() => counter += 1, 1000);

	onDestroy(() => clearInterval(interval));
</script>
```

While it's important to call lifecycle functions during the component's initialisation, it doesn't matter *where* you call them from. So if we wanted, we could abstract the interval logic into a helper function in `utils.js`...

```
import { onDestroy } from 'svelte';

export function onInterval(callback, milliseconds) {
	const interval = setInterval(callback, milliseconds);

	onDestroy(() => {
		clearInterval(interval);
	});
}
```

...and import it into our component:

```
<script>
	import { onInterval } from './utils.js';

	let counter = 0;
	onInterval(() => counter += 1, 1000);
</script>
```

Open and close the timer a few times and make sure the counter keeps ticking and the CPU load increases. This is due to a memory leak as the previous timers are not deleted. Don't forget to refresh the page before solving the example.