Watch: 🐛 **A refactorizar: solucionamos un bug gracias a let!**

Tienes el código de este vídeo en GitHub.

Podemos ver un ejemplo en el que la ausencia de block scope de `var` nos provoca un bug:

```
var contentCounters = document.querySelectorAll(".js-count-
content");

for (var i = 0; i < contentCounters.length; ++i) {
  var counter = contentCounters[i]
  var form_field = counter.parentElement.querySelector(
    ".js-form-control"
  );
  var char_counter_container = counter.querySelector(".js-
count-chars");

  char_counter_container.innerHTML =
countChars(form_field.value);

  form_field.addEventListener("keyup", function () {
```

```
    char_counter_container.innerHTML =
countChars(form_field.value);
  });
}
```

Este código se ejecuta cuando carga la página, pero el callback del event listener no se ejecutará hasta que el usuario escriba en el input. En ese momento, como `char_counter_container` y `form_field` no estan definidas en el scope de la función, se irán a buscar al scope superior. Al ser un loop, las variables `var` se han sobreescrito y harán siempre referencia a la última iteración del loop `for`.

Si cambiamos las variables a `let` o `const` vemos como el bug desaparece, ya que el block scope hace que en el momento en el que se ejecute el callback, el scope siga siendo el de la iteración en la que se generó el listener.