Article wrapper

Callouts using `co`:

```
(let  loopvar  ((count 1))
   (if (> count 10)
     #t
    ( loopvar  (+ count 1))))

(let  loopvar  ((count 1))
   (if (> count 10)
     #t
    (11loopvar 12(+ count 1))))

(let 13loopvar 14((count 1))
  15(if (> count 10)
    16#t
    (17loopvar 18(+ count 1))))

(let 19loopvar 20((count 1))
  21(if (> count 10)
    22#t
    (23loopvar 24(+ count 1))))
```

This variable controls the loop. It is declared without an initial value, immediately after the `let` operand.

Any number of additional local variables can be defined after the loop variable, just as they can in any other `let` expression.

If you ever want the loop to end, you have to put some sort of a test in it.

This is the value that will be returned.

Note that you iterate the loop by using the loop variable as if it was a function name.

The arguments to this function are the values that you want the local variables declared in to have in the next iteration.

This variable controls the loop. It is declared without an initial value, immediately after the `let` operand.

Any number of additional local variables can be defined after the loop variable, just as they can in any other `let` expression.

If you ever want the loop to end, you have to put some sort of a test in it.

This is the value that will be returned.

11 Note that you iterate the loop by using the loop variable as if it was a function name.

12 The arguments to this function are the values that you want the local variables declared in to have in the next iteration.

13 This variable controls the loop. It is declared without an initial value, immediately after the `let` operand.

14 Any number of additional local variables can be defined after the loop variable, just as they can in any other `let` expression.

15 If you ever want the loop to end, you have to put some sort of a test in it.

16 This is the value that will be returned.

17 Note that you iterate the loop by using the loop variable as if it was a function name.

18 The arguments to this function are the values that you want the local variables declared in to have in the next iteration.

19 This variable controls the loop. It is declared without an initial value, immediately after the `let` operand.

20 Any number of additional local variables can be defined after the loop variable, just as they can in any other `let` expression.

21 If you ever want the loop to end, you have to put some sort of a test in it.

22 This is the value that will be returned.

23 Note that you iterate the loop by using the loop variable as if it was a function name.

24 The arguments to this function are the values that you want the local variables declared in to have in the next iteration.