**define-syntax live demo**

**(define-syntax my-let**

**(syntax-rules ()**

**[(\_ ((x v) ...) e1 e2 ...)**

**((lambda (x ...) e1 e2 ...)**

**v ...)]))**

**(define-syntax my-if**

**(syntax-rules (then else)**

**((\_ e1 then e2)**

**(if e1 e2))**

**((\_ e1 then e2 else e3)**

**(if e1 e2 e3))))**

**(define-syntax ++**

**(syntax-rules ()**

**((\_ x)**

**(begin (set! x (+ 1 x))**

**x))))**

**(define-syntax ++-post**

**(syntax-rules ()**

**((\_ x)**

**(let ([old-value x])**

**(set! x (+ 1 x))**

**old-value))))**

**(define-syntax for**

**(syntax-rules (:)**

**((\_ ((init ...) : test : update ...) body ...)**

**(begin**

**init ...**

**(let loop ()**

**(if test**

**(begin**

**body ...**

**update ...**

**(loop))))))**

**((\_ (init : test : update ...) body ...)**

**(begin**

**init**

**(let loop ()**

**(if test**

**(begin**

**body ...**

**update ...**

**(loop))))))))**

**'(for (((define i 0) (define j 1)) :**

**(< i 12) :**

**(++ i) (set! j (\* 2 j)))**

**(display i)**

**(display " ")**

**(display j)**

**(newline))**

**(define-syntax my-and**

**(syntax-rules ()**

**((\_) #t)**

**((\_ exp) exp)**

**((\_ exp exp2 ...)**

**(if exp**

**(my-and exp2 ...)**

**#f))**