```
;; Disklavier Studies 1

;; generation of pitches 'to create a surface'

(setq material1 (list-a-scale '-\e 127))

(gen-random-successive 0.1 24 material1) ; this will give you some repetition
; (-u r -w -f f m -x -r -x n -o -v -q -x e c q -d -c -q r y q -b)

(gen-random-unique 0.1 24 material1) ; this is an absolutely unique list
; (p w j -p -m -u -w -f f m -r n -o -v -x e c -d -c -q r y q -b)

;; symbols (notes)
;; this expression generates and collect 12 unique collections of symbols

(setq material1-12 (gen-collect 0.6 12 :list
                          '(gen-random-unique nil 24 material1)))


#| ;; output of material1-12

((-j -c -p j m e c t -t -m h -l k d b -u -w v w i -h a -i x)
(-g -q -m -w s m d -s q n -r v o w -e -x t -p h r -k -t f a)
(-g -t p -m -b -f -v -e u s -i n e -r f a r h j t q -p -d -o)
(-l p -e b f -w g -b d -k t o i q -n -u s e w -d c -x x -j)
(y c n l -n b m o f i -l -p k -s h t r -k -u g -j j -i -b)
(-l -t o f -o -b a -w u x b -p e -h m q -x -k -e l -v i -m -f)
(m -h -i -t -r -n -g -j f x s -x t v -w -d i u p -c b -o j -m)
(-q g -c r y b -v -m -t j c l u a e -o -r o x -b -j -u i w)
(v -n o d -k s -o h -i -e -p p j q -t f i -v w -l -g k u -c)
(y c -o -r -j -d -u -k h b -m -n i -x -g d -t k v p j n -b a)
(-i e -h w -r k -n o -e -v s m -b g -o -p p -j -g n q t v -d)
(-d -c k v m -s s u r e t g o l -j -b -k -t p f -v y h -n))


;; run this expression to look at the symbols chosen in a scalic sequence

(mapcar 'sort< material1-12)

((-w -u -t -p -m -l -j -i -h -c a b c d e h i j k m t v w x)
(-x -w -t -s -r -q -p -m -k -g -e a d f h m n o q r s t v w)
 . . . etc

|#

;; tonality scheme
;; I know you don't want 'phrases' as such . . .
;; but this tonality-shift let's you hear each collection clearly
```

```
(setq with-tonality-shift (interpolate-tonality 12
                                          (activate-tonality (chromatic f# 2))
                                          (activate-tonality (chromatic f# 4))))

#|
((c 3 c# 3 d 3 d# 3 e 3 f 3 f# 3 g 3 g# 3 a 3 a# 3 b 3)
 (d 3 d# 3 e 3 f 3 f# 3 g 3 g# 3 a 3 a# 3 b 3 c 4 c# 4)
 (e 3 f 3 f# 3 g 3 g# 3 a 3 a# 3 b 3 . . . .

. . . . . .  d 5 d# 5 e 5 f 5 f# 5 g 5 g# 5 a 5)
(c 5 c# 5 d 5 d# 5 e 5 f 5 f# 5 g 5 g# 5 a 5 a# 5 b 5))

|#

;; creates an articulation surface of staccato and resonance

(setq dur-lis '(1/32 1/32 1/36 1/40 1/44 1/48 1/52 1/56 1/60 1/64 1/64 1/64))

(setq dur (append (gen-repeat 3 '((1/64)))
              (list (gen-repeat 3  dur-lis))
              (gen-collect 0.1 6 :list
                           '(symbol-shuffle
                             (change-length :times 2  dur-lis :ratio)))))


;; score


(def-tonality
dk1 with-tonality-shift
)


(def-symbol
dk1 material1-12
)

(def-length
 dk1 '(1/32)
)

(def-duration
  dk1 dur
)

(def-velocity
dk1 (symbol-to-velocity 40 80 2 material1-12)
)

(def-expression ; this adds a little roughness to the output
```

```
dk1 ((legato 70 30 0.4) (humanize -2 3 0.4) (velocity 20 0.1))
)


(def-zone
 dk1 (gen-repeat 12 '(3/4))
)

(def-channel
dk1  1
)

(def-program gm-sound-set
dk1  1

)

(def-tempo 80)

(compile-instrument-p "ccl;output:" "dk-test1"
dk1
)









;; Disklavier Studies 2

;; generation of pitches 'to create a surface' with a 'target' note of
;; minimum duration in each group from which durations of increasing size flow and ebb

(setq material1 (list-a-scale '-x 48))

(gen-random-unique 0.1 24 material1)
; (p w j -p -m -u -w -f f m -r n -o -v -x e c -d -c -q r y q -b)

;; symbols (notes)
(setq material1-12 (gen-collect 0.6 12 :list
                          '(gen-random-unique nil 24 material1)))
```

```
;; tonality scheme
(setq with-tonality-shift (interpolate-tonality 12
                                          (activate-tonality (chromatic c 3))
                                          (activate-tonality (chromatic c 5))))

;; selects a 'target' symbol from each list of 12 unique pitch selections

(setq lisy
      (flatten (gen-process-list '(p-select x y)
                (symbol-shuffle (list-a-scale 0 23) 0.1) material1-12)))
; >  ((a) (w) (f) (-u) (i) (a) (-r) (-j) (s) (c) (w) (-d))

;; identifies the position in each list of the selected symbol

(setq lisi
      (flatten (gen-process-list '(e-position x y) lisy material1-12)))
; > (21 13 14 15 9 6 4 20 5 1 3 0)

;; adding a randomly-generated chord (using Robert Rowe's Cypher algorithm) to the 'target' note
;; if only so you can hear exactly where the 'target' symbol is.

(setq material1-12x
      (gen-process-list
       '(p-replace nil x
         (cypher-chorder -13 6 27 (list (integer-to-symbol x)) nil)  y) lisi material1-12))




#| ;; output of material1-12x

> ((-j -c -p j m e c t -t -m h -l k d b -u -w v w i -h vqm -i x)
(-g -q -m -w s m d -s q n -r v o nktz -e -x t -p h r -k -t f a)
(-g -t p -m -b -f -v -e u s -i n e -r opa a r h j t q -p -d -o)
(-l p -e b f -w g -b d -k t o i q -n pduk s e w -d c -x x -j)
(y c n l -n b m o f jbos -l -p k -s h t r -k -u g -j j -i -b)
(-l -t o f -o -b g-diu -w u x b -p e -h m q -x -k -e l -v i -m -f)
(m -h -i -t e-cjm -n -g -j f x s -x t v -w -d i u p -c b -o j -m)
(-q g -c r y b -v -m -t j c l u a e -o -r o x -b upwf -u i w)
(v -n o d -k f-cjh -o h -i -e -p p j q -t f i -v w -l -g k u -c)
(y b-dfc -o -r -j -d -u -k h b -m -n i -x -g d -t k v p j n -b a)
(-i e -h d-bhr -r k -n o -e -v s m -b g -o -p p -j -g n q t v -d)
(a-fdu -c k v m -s s u r e t g o l -j -b -k -t p f -v y h -n))
|#


;; note-lengths
;; makes a list of ticks between 1/64 and just over an 1/8
```

```
(setq lisl
      (change-length :add 30 (list-a-scale 0 24 :scr 10)))
; (30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260)

;; this mechanism lines up symbols and tick values

(setq tick-lengths1
 (gen-process-list '(symbol-scroll x y) lisi (build-list  lisl 12)))

#|
> ((60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50)
(140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130)
(130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120)
(120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110)
(180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170)
(210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200)
(230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220)
(70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60)
(220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210)
(260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250)
(240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230)
(30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260))
|#

;; here's a different approach . . . to produce a flow and ebb from the target

(setq lis2
      (append (gen-palindrome (change-length :add 30 (list-a-scale 0 12 :scr 10))) '(29)))
; > (30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30)

;; this mechanism lines up symbols and tick values

(setq tick-lengths2
 (gen-process-list '(symbol-scroll x y) lisi (build-list  lis2 12)))


#|
> ((60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50)
(140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130)
(130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120)
(120 130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110)
(100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110)
(70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80)
(50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60)
(70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60)
(60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70)
(29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30)
(40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50)
(30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30 29))
```

```
|#

;; dynamics

(setq dynamics1
      (symbol-divide 24 nil nil (vector-to-list (symbol-to-velocity 20 60 2 material1-12))))

;; this makes the 'target' chord  play at maximum velocity

(setq dynamics2 (gen-process-list '(p-replace nil x (get-random 70 100) y) lisi dynamics1))


;; durations / articulations
;; this places a random duration on the target chord

(setq tick-durations1
      (gen-process-list
       '(symbol-scroll x y) lisi
       (build-list (p-replace nil 'first (get-random 30 240) lis2) 12)))
;; score-template


(def-tonality
dk1 with-tonality-shift
)

(def-symbol
dk1 material1-12x
)

(def-length
 dk1  tick-lengths2
)

(def-duration
dk1  tick-durations1
)

(def-velocity
dk1 dynamics2
)

(def-expression
dk1 ((legato 70 30 0.4) (humanize -2 3 0.4) (velocity 20 0.1))
)

(def-zone
 dk1  (gen-repeat 12 (list (make-zone (car tick-lengths2))))
)
```

```
(def-channel
dk1  1
)

(def-program gm-sound-set
dk1  1

)

(def-tempo 80)

(compile-instrument-p "ccl;output:" "dk-test2"
dk1
)


;; Disklavier Studies 3

;; generation of pitches 'to create a surface' with a 'target' note of minimum
; duration in each group from which first: durations of increasing size flow and ebb
; and then second; durations are shuffled to produce a rhythmic articulation in the
; time-space between the 'target' notes

(setq material1 (list-a-scale '-x 48))

(gen-random-unique 0.1 24 material1)
; (p w j -p -m -u -w -f f m -r n -o -v -x e c -d -c -q r y q -b)

(setq material1-12 (gen-collect 0.6 12 :list
                          '(gen-random-unique nil 24 material1)))


;; tonality scheme

(setq with-tonality-shift (interpolate-tonality 12
                                          (activate-tonality (chromatic f# 3))
                                          (activate-tonality (chromatic f# 5))))

;; selects a 'target' symbol from each list of 12 unique pitch selections

(setq lisy
      (flatten (gen-process-list '(p-select x y)
               (symbol-shuffle (list-a-scale 0 23) 0.1) material1-12)))
; >  ((a) (w) (f) (-u) (i) (a) (-r) (-j) (s) (c) (w) (-d))

;; identifies the position in each list of the selected symbol

(setq lisi
      (flatten (gen-process-list '(e-position x y) lisy material1-12)))
```

```
; > (21 13 14 15 9 6 4 20 5 1 3 0)

;; adding a randomly-generated chord (using Robert Rowe's Cypher algorithm) to the 'target' note
;; if only so you can hear exactly where the 'target' symbol is.

(setq material1-12x
      (gen-process-list
       '(p-replace nil x
         (cypher-chorder -13 6 27 (list (integer-to-symbol x)) nil)  y) lisi material1-12))

(setq material1-12y
      (gen-process-list
       '(p-replace nil x
         (cypher-chorder -13 6 27 (list (integer-to-symbol x)) nil)  y) lisi material1-12))
#|
> ((-j -c -p j m e c t -t -m h -l k d b -u -w v w i -h vqm -i x)
(-g -q -m -w s m d -s q n -r v o nktz -e -x t -p h r -k -t f a)
(-g -t p -m -b -f -v -e u s -i n e -r opa a r h j t q -p -d -o)
(-l p -e b f -w g -b d -k t o i q -n pduk s e w -d c -x x -j)
(y c n l -n b m o f jbos -l -p k -s h t r -k -u g -j j -i -b)
(-l -t o f -o -b g-diu -w u x b -p e -h m q -x -k -e l -v i -m -f)
(m -h -i -t e-cjm -n -g -j f x s -x t v -w -d i u p -c b -o j -m)
(-q g -c r y b -v -m -t j c l u a e -o -r o x -b upwf -u i w)
(v -n o d -k f-cjh -o h -i -e -p p j q -t f i -v w -l -g k u -c)
(y b-dfc -o -r -j -d -u -k h b -m -n i -x -g d -t k v p j n -b a)
(-i e -h d-bhr -r k -n o -e -v s m -b g -o -p p -j -g n q t v -d)
(a-fdu -c k v m -s s u r e t g o l -j -b -k -t p f -v y h -n))
|#

;; note-lengths
;; makes a list of ticks between 1/64 and just over an 1/8

(setq lisl
      (change-length :add 30 (list-a-scale 0 24 :scr 10)))
; (30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260)

;; this mechanism lines up symbols and tick values

(setq tick-lengths1
 (gen-process-list '(symbol-scroll x y) lisi (build-list  lisl 12)))

#|
> ((60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50)
(140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130)
(130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120)
(120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110)
(180 190 200 210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170)
(210 220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200)
(230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220)
(70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 30 40 50 60)
```

```
(220 230 240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210)
(260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250)
(240 250 260 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230)
(30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260))
|#

;; here's a different approach . . . to produce a flow and ebb from the target

(setq lis2
      (append (gen-palindrome (change-length :add 30 (list-a-scale 0 12 :scr 10))) '(29)))
; > (30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30)

;; this produces a rhythmic articulation in the time-space between the 'target'

(setq lis3 (gen-collect 0.1 12 :list '(flatten (do-section '(= x) '(symbol-shuffle x)
                       (symbol-divide '(3 21) nil nil (symbol-scroll 2 lis2))))))

;; this mechanism lines up symbols and tick values

(setq tick-lengths2
 (gen-process-list '(symbol-scroll x y) lisi (build-list  lis2 12)))

(setq tick-lengths3
      (gen-process-list '(symbol-scroll x y) lisi lis3))

#|
; output of tick-lengths2  . . . with flow and ebb

> ((60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50)
(140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130)
(130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120)
(120 130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110)
(100 90 80 70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110)
(70 60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80)
(50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60)
(70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30 29 30 40 50 60)
(60 50 40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70)
(29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30)
(40 30 29 30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50)
(30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30 29))
```

```
; output of tick-lengths3 " . . . and the durations converge rhythmically"

> ((80 40 90 100 120 50 110 80 140 50 100 110 70 120 130 90 60 60 40 70 130 30 29 30)
(70 90 50 110 120 40 100 70 60 40 130 110 120 30 29 30 130 80 100 140 80 90 50 60)
(70 80 140 130 110 90 60 120 50 120 130 100 60 110 30 29 30 50 40 100 90 40 70 80)
(80 40 80 50 70 110 70 100 140 90 120 100 120 60 50 30 29 30 40 60 90 130 110 130)
(60 100 80 80 120 70 100 60 40 30 29 30 110 70 90 110 130 90 50 50 120 40 130 140)
(50 40 120 140 80 60 30 29 30 60 70 100 40 110 50 80 100 130 110 70 90 130 90 120)
(140 50 80 50 30 29 30 70 120 120 130 100 110 70 90 100 90 40 80 110 60 40 60 130)
(60 80 100 110 140 50 40 120 110 130 70 90 50 100 70 90 60 130 80 120 30 29 30 40)
(110 70 80 60 140 30 29 30 40 120 90 90 130 80 40 100 60 120 130 110 70 50 50 100)
 (130 30 29 30 100 110 120 50 50 60 100 90 70 80 130 120 140 110 70 90 60 40 80 40)
(120 50 70 30 29 30 120 100 40 70 90 80 90 40 80 50 60 60 100 110 130 110 130 140)
(30 29 30 50 100 60 90 110 120 60 130 120 40 50 80 90 100 130 40 70 110 70 140 80))

|#

;; durations/articulations
;; this places a random duration on the target chord

(setq tick-durations1
      (gen-process-list
       '(symbol-scroll x y) lisi
       (build-list (p-replace nil 'first (get-random 30 240) lis2) 12)))

;; dynamcis

(setq dynamics1
      (symbol-divide 24 nil nil (vector-to-list (symbol-to-velocity 20 60 2 material1-12))))

;; this makes the 'target' chord  play at maximum velocity

(setq dynamics2 (gen-process-list '(p-replace nil x (get-random 70 100) y) lisi dynamics1))









;; score-template

(def-tonality
dk1 (append (gen-palindrome with-tonality-shift)
            (transpose-tonality 6 (last with-tonality-shift)))
```

```
)

(def-symbol
dk1 (append material1-12x  material1-12y)
)

(def-length
 dk1  (append tick-lengths2 tick-lengths3)
)

(def-duration
dk1 (append tick-durations1 tick-durations1)
)

(def-velocity
dk1 (append dynamics1  dynamics2)
)

(def-expression
dk1 ((legato 70 30 0.4) (humanize -2 3 0.4) (velocity 20 0.1))
)

(def-zone
 dk1  (gen-repeat 24 (list (make-zone (car tick-lengths2))))
)

(def-channel
dk1  1
)

(def-program gm-sound-set
dk1  1
)

(def-tempo 80)

(compile-instrument-p "ccl;output:" "dk-test3"
dk1
)


;; Disklavier Studies 4

 #|

Generation of pitches:

a) 'to create a surface of notes between pppp and p';

b) having a two sequences of 'target' notes of minimum duration in each group
```

```
 One:
   i)  with dynamics between mp and f;
   ii) having durations of increasing size flow and ebb between target notes


 Two:
   i) with durations are shuffled to produce a rhythmic articulation in the
  time-space between 'target' notes.

d) In the section where durations are shuffled the target note collects
   together all previously sounded target notes.
   i) this group is triggered as each target note is reached
   ii) the content of the group is always rotated to start at the
       appropriate 'next' note.

|#


(setq material1 (list-a-scale '-x 48))

(gen-random-unique 0.1 24 material1)
; (p w j -p -m -u -w -f f m -r n -o -v -x e c -d -c -q r y q -b)

(setq material1-12 (gen-collect 0.6 12 :list
                            '(gen-random-unique nil 24 material1)))

; > complete output below




#|

((-j -c -p j m e c t -t -m h -l k d b -u -w v w i -h a -i x)
(-g -q -m -w s m d -s q n -r v o w -e -x t -p h r -k -t f a)
(-g -t p -m -b -f -v -e u s -i n e -r f a r h j t q -p -d -o)
(-l p -e b f -w g -b d -k t o i q -n -u s e w -d c -x x -j)
(y c n l -n b m o f i -l -p k -s h t r -k -u g -j j -i -b)
(-l -t o f -o -b a -w u x b -p e -h m q -x -k -e l -v i -m -f)
(m -h -i -t -r -n -g -j f x s -x t v -w -d i u p -c b -o j -m)
(-q g -c r y b -v -m -t j c l u a e -o -r o x -b -j -u i w)
(v -n o d -k s -o h -i -e -p p j q -t f i -v w -l -g k u -c)
(y c -o -r -j -d -u -k h b -m -n i -x -g d -t k v p j n -b a)
(-i e -h w -r k -n o -e -v s m -b g -o -p p -j -g n q t v -d)
(-d -c k v m -s s u r e t g o l -j -b -k -t p f -v y h -n))
```

```
|#

;; tonality scheme

(setq with-tonality-shift (interpolate-tonality 12
                                          (activate-tonality (chromatic c 3))
                                          (activate-tonality (chromatic c 5))))

#|
> ((c 3 c# 3 d 3 d# 3 e 3 f 3 f# 3 g 3 g# 3 a 3 a# 3 b 3)
 (d 3 d# 3 e 3 f 3 f# 3 g 3 g# 3 a 3 a# 3 b 3 c 4 c# 4)
 (e 3 f 3 f# 3 g 3 g# 3 a 3 a# 3 b 3 . . . .

. . . . . .  d 5 d# 5 e 5 f 5 f# 5 g 5 g# 5 a 5)
(c 5 c# 5 d 5 d# 5 e 5 f 5 f# 5 g 5 g# 5 a 5 a# 5 b 5))
|#


;; selecting a 'target' note (symbol) from each list of 12 unique pitch selections

(setq lisy
      (flatten (gen-process-list '(p-select x y)
               (symbol-shuffle (list-a-scale 0 23) 0.1) material1-12)))
; >  ((a) (w) (f) (-u) (i) (a) (-r) (-j) (s) (c) (w) (-d))

(setq lisx (gen-process-list  '(remove x y) lisy material1-12))
; > ((-j -c -p j m e c t -t -m h -l k d b -u -w v w i -h -i x) . . .target note 'a is removed

(setq lisz (gen-process-list '(filter-delete x y) lisx material1-12))
; > ((= = = = = = = = = = = = = = = = = = = = = = a = =) . . . lists of target notes created

(setq material1-12z (gen-process-list  '(filter-delete x y) lisy material1-12))

; ((-j -c -p j m e c t -t -m h -l k d b -u -w v w i -h = -i x)
; (-g -q -m -w s m d -s q n -r v o = -e -x t -p h r -k -t f a) . . . .
; . . . the original 'material' list is now without the target note


;; creating a new series of lists based on the starting point of each target note

(setq n-len
      (mapcar 'length
               (create-lists
                (flatten (mapcar 'create-lists
                                 (gen-process-list
                                  '(filter-delete x y) lisy material1-12))))))
; > (21 16 25 25 18 21 22 40 9 20 26 21 24)

(setq r-lis (gen-process '(build-list y x ) n-len  '= :list))
; ((= = = = = = = = = = = = = = = = = = = = =) (= = = = = = = = = = = = = = = =) . . .
```

```
;; each new list is filled with a rotation of the target group of notes

(setq target-group-list
      (append (list (car r-lis))
              (gen-process-list '(replace x y) (cdr r-lis) (rotate-f lisy))))

#| ; note the different list lengths
>
((= = = = = = = = = = = = = = = = = = = = =)
(a w f -u i a -r -j s c w -d = = = =)
(w f -u i a -r -j s c w -d a = = = = = = = = = = = = =)
 (f -u i a -r -j s c w -d a w = = = = = = = = = = = = =)
(-u i a -r -j s c w -d a w f = = = = = =)
(i a -r -j s c w -d a w f -u = = = = = = = = =)
(a -r -j s c w -d a w f -u i = = = = = = = = = =)
(-r -j s c w -d a w f -u i a = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =)
(-j s c w -d a w f -u)
(s c w -d a w f -u i a -r -j = = = = = = = =)
(c w -d a w f -u i a -r -j s = = = = = = = = = = = = =)
(w -d a w f -u i a -r -j s c = = = = = = = = =)
(-d a w f -u i a -r -j s c w = = = = = = = = = = = =))
|#


(init-rnd 0.1) ; sets the random seed as <get-random> has no seed parameter

(setq tgla (do-section :all '(symbol-transpose (get-random -7 7) x) (but-last target-group-list))
      tglb (last target-group-list)
      tgla/b (append tgla tglb))


;;  creating note-lengths
;; identifying the position in each list of the selected symbol

(setq lisi
      (flatten (gen-process-list '(e-position x y)  lisy material1-12)))
; > (21 13 14 15 9 6 4 20 5 1 3 0)


;; producing a rhythmic flow and ebb from the target using tick values
;; how to work out the default resolution in ticks. . . (get-tick '1/4) > 480


(setq lis2
      (append (gen-palindrome (change-length :add 30 (list-a-scale 0 12 :scr 10))) '(29)))
; > (30 40 50 60 70 80 90 100 110 120 130 140 130 120 110 100 90 80 70 60 50 40 30)


;; This produces a rhythmic articulation in the time-space between the 'target' note
```

```
(setq lis3 (gen-collect 0.1 12 :list '(flatten (do-section '(= x) '(symbol-shuffle x)
                       (symbol-divide '(3 21) nil nil (symbol-scroll 2 lis2))))))

;; the mechanism below lines up symbols and tick values

(setq tick-lengths2
 (gen-process-list '(symbol-scroll x y) lisi (build-list  lis2 12)))

(setq tick-lengths3
      (gen-process-list '(symbol-scroll x y) lisi lis3))

(setq tick-lengths3a
      (symbol-divide n-len nil nil (flatten tick-lengths3)))




;; dynamics
;; this makes the 'surface' play between pppp and p

(setq dynamics1
      (symbol-divide 24 nil nil (vector-to-list (symbol-to-velocity 20 45 2 material1-12))))


;; this makes the 'target' note play between mp and f

(setq dynamics2 (gen-process-list '(p-replace nil x (get-random 65 110) y) lisi dynamics1))

;; this 'on the fly' function <select-dynamics> produces a random mix of
;  dynamic patterns within the target-group-list

(defun select-dynamics (lis)
      (let ((selection (pick-random '(:this :that :other ))))
         (case selection
           (:this
              (gen-cresc 60 100 (length lis)))
           (:that
              (gen-dim 90 60 (length lis)))
               (:other
                (gen-cresc-dim 60 90 (length lis))))))

(setq tgl-dynamics (do-section :all  '(select-dynamics x) target-group-list))

;; revised existing system function if <def-duration> activated

(defun resonance-maker-2 (val symbol-lists)
  "simulates a sustain pedal effect but allowing for ticks in the val parameter"
```

```
  (do-quietly
    (mapcar (function (lambda (x) (change-length :times val x)))
            (mapcar (function (lambda (x) (g-integer x 1)))
                    (mapcar 'list (mapcar 'length symbol-lists))))))
```

```
;; score-template


(def-tonality
dk1 (append (gen-palindrome with-tonality-shift)
            (transpose-tonality 6 (last with-tonality-shift)))
dk2 (append with-tonality-shift
            (gen-repeat 13 (transpose-tonality 6 (last with-tonality-shift))))
)

(def-symbol
dk1 (append material1-12z  material1-12z)
dk2 (append lisz  tgla/b)
)

(def-length
 dk1  (append tick-lengths2 tick-lengths3)
 dk2  (append tick-lengths2 tick-lengths3a)
)

(def-velocity
dk1 (append dynamics1  dynamics1)
dk2 (append dynamics2  tgl-dynamics)
)

(def-expression
default ((legato 70 30 0.4) (humanize -2 3 0.4) (velocity 20 0.1))
)

(def-duration ; optional!
dk2 (resonance-maker-2 '30 (length-of dk2))
)
```

```
(def-zone
 dk1 (gen-repeat 24 (list (make-zone (car tick-lengths2))))
 dk2  (append (gen-repeat 12 (list (make-zone (car tick-lengths2))))
              (mapcar 'make-zone tick-lengths3a))
)

(def-channel
dk1 1
dk2 2
)

(def-program gm-sound-set
default  1
)

(def-tempo 80)

(compile-instrument-p "ccl;output:" "dk-test4"
dk1
dk2
)
```