CheatSheet

MyGloss1.hs

```haskell
import Graphics.Gloss

fib :: [Float]
fib = 1:1:zipWith (+) fib (tail fib)

posfib' :: Num t => t -> [t] -> [(t, t)]
posfib' c (x:xs) = (c, x) : posfib' (c + x + head xs) xs
posfib' _ _ = error "posfib'"

posfib :: Num t => [t] -> [(t, t)]
posfib f = posfib' 0 f

posCircle :: Float -> (Float, Float) -> Picture
posCircle r (x, y) = Translate x y $ Circle r

mypic :: Picture
mypic = Pictures $ take 20 $ zipWith posCircle fib $ posfib fib

main :: IO ()
main = display d white mypic
  where d = InWindow "Nice Window" (1024 `div` 2, 768 `div` 2) (10, 10)
```

MyGloss2.hs

```haskell
import Graphics.Gloss

fib :: [Float]
fib = 1:1:zipWith (+) fib (tail fib)

data Way = WLeft | WRight | WUp | WDown
next :: Way -> Way
next WLeft  = WDown
next WDown  = WRight
next WRight = WUp
next WUp    = WLeft

jumpTo :: Num t => Way -> t -> t -> (t, t)
jumpTo WLeft a b  = (-b - a, -b + a)
jumpTo WDown a b  = ( b - a, -b - a)
jumpTo WRight a b = ( b + a,  b - a)
jumpTo WUp a b    = (-b + a,  b + a)

posFib' :: Num t => (t, t) -> Way -> [t] -> [(t, t)]
posFib' c w (x:xs) = c' : posFib' c' (next w) xs
  where
    tplus a b = let (f, s) = unzip [a, b] in (sum f, sum s)
    c' = c `tplus` jumpTo w x (head xs)
posFib' _ _ _ = error "posFib'"

posFib :: Num t => [t] -> [(t, t)]
posFib f = posFib' (0, 0) WDown (0:f)

posCircle :: Float -> (Float, Float) -> Picture
posCircle r (x, y) = Translate x y $ Circle r

mypic :: Picture
mypic = Pictures $ take 20 $ zipWith posCircle r $ posFib r
  where r = fmap (*5) fib

main :: IO ()
main = display d white mypic
  where d = InWindow "Nice Window" (1024 `div` 2, 768 `div` 2) (10, 10)
```

MyGloss3.hs

```diff
--- MyGloss2.hs 2012-05-26 20:26:53.412290962 +0900
+++ MyGloss3.hs 2012-05-26 20:23:55.131406913 +0900
@@ -29,10 +29,12 @@
 posCircle :: Float -> (Float, Float) -> Picture
 posCircle r (x, y) = Translate x y $ Circle r

-mypic :: Picture
-mypic = Pictures $ take 20 $ zipWith posCircle r $ posFib r
-  where r = fmap (*5) fib
+mypic :: Float -> Picture
+mypic t = Scale s s $ Pictures $ take n $ zipWith posCircle r $ posFib r
+  where s = 10 / (1.4 ** t)
+        r = fmap (*5) fib
+        n = truncate t

 main :: IO ()
-main = display d white mypic
+main = animate d white mypic
   where d = InWindow "Nice Window" (1024 `div` 2, 768 `div` 2) (10, 10)
```