```
 1: # This is an enhanced version of the chain (catenary curve) example
 2: # discussed in Chapter 5 of the book Object-Oriented Programming in Python
 3: # Authors: Jason Fritts, Michael Goldwasser
 4:
 5: from cs1graphics import *
 6: from math import sqrt
 7:
 8: #************************
 9: #  Define constants
10: #************************
11:
12: # example chain length, number of links, link length, etc.
13: displaySpeed        = 1.0           # larger for faster display, and vice versa
14: numLinks            = 50            # number of chain links
15: restingLength       = 21.0          # initial length of each link (in pixels)
16: totalSeparation     = 630.0         # distance between chain ends (in pixels)
17:
18: gravityConstant     = ( 1.0 / 32.2 ) * (totalSeparation / 600)
19: elasticityConstant  = 1.0 / totalSeparation
20: speed               = displaySpeed * (numLinks / 20)
21: epsilon             = sqrt(speed) / max(totalSeparation,400)
22:
23:
24: #************************
25: #  Define functions
26: #************************
27:
28: #### convenient function for adding up to three (x,y) tuples
29: def combine(A, B, C=(0,0) ):
30:     return (A[0] + B[0] + C[0], A[1] + B[1] + C[1])
31:
32: #### function returns a tuple representing the force being
33: #### exerted upon point A due to the link joining A to B.
34: def calcForce(A, B):
35:     dX = (B[0] - A[0])
36:     dY = (B[1] - A[1])
37:     distance = sqrt(dX * dX + dY * dY)
38:
39:     forceFactor = 0
40:     if distance > restingLength:                  # link being stretched
41:         stretch = distance - restingLength
42:         forceFactor = stretch * elasticityConstant
43:
44:     return (forceFactor * dX, forceFactor * dY)   # returning a tuple
45:
46: #### function to alter the graphical path and refresh canvas
47: def drawChain(chainData, chainPath, theCanvas):
48:     for k in range(len(chainData)):
49:         chainPath.setPoint(Point(chainData[k][0], chainData[k][1]), k)
50:     theCanvas.refresh()
51:
52: #************************
53: #  Main program
54: #************************
55:
56: # initialize the chain; one end at (0,0) other at (totalSeparation,0)
57: chain = []
58: for k in range(numLinks + 1):
59:     X = totalSeparation * k / numLinks
60:     chain.append( (X, 0.0) )        # add new position
61:
62: # initialize the graphics
63: paper = Canvas(totalSeparation, totalSeparation)
64: paper.setAutoRefresh(False)
65:
66: curve = Path()
67: for p in chain:
68:     curve.addPoint(Point(p[0], p[1]))
69: paper.add(curve)
70:
71: graphicsCounter = int(25 * speed)   # we will only draw some iterations
```

```
 72:
 73: # as long as forces are not (sufficiently) in equilibrium, adjust
 74: #    chain link positions and re-draw
 75: somethingMoved = True               # force loop to start
 76: while somethingMoved:
 77:     somethingMoved = False          # default for new iteration
 78:     oldChain = list(chain)          # record a copy of the data
 79:
 80:     # examine forces being applied to a chain link, and
 81:     #    adjust link position if not at equilibrium
 82:     for k in range(1, numLinks):
 83:         gravForce = (0, gravityConstant)                    # downward force
 84:         leftForce = calcForce(oldChain[k], oldChain[k-1])
 85:         rightForce = calcForce(oldChain[k], oldChain[k+1])
 86:         adjust = combine(gravForce, leftForce, rightForce)
 87:         chain[k] = combine(oldChain[k], adjust)
 88:
 89:         if abs(adjust[0]) > epsilon or abs(adjust[1]) > epsilon:
 90:             somethingMoved = True
 91:
 92:     # only draw intermittent results, otherwise display takes too long
 93:     graphicsCounter -= 1
 94:     if graphicsCounter == 0:
 95:         drawChain(chain, curve, paper)
 96:         graphicsCounter = int(25 * speed)
 97:
 98: # display final result with line emphasized (bold)
 99: curve.setBorderWidth(2)
100: drawChain(chain, curve, paper)
```