

```
1 #py
2 #
3
4 # This is an example which shows how to complete a MatDeck document in Python
5 # code. Using this way, the MatDeck Code Editor serves as the Python editor.
6
7 """
8 Animated Towers of Hanoi using Tk with optional bitmap file in background.
9 Usage: hanoi.py [n [bitmapfile]]
10
11 n is the number of pieces to animate; default is 4, maximum 15.
12
13 The bitmap file can be any X11 bitmap file (look in /usr/include/X11/bitmaps
14 for
15 samples); it is displayed as the background of the animation. Default is no
16 bitmap.
17 """
18
19 from tkinter import Tk, Canvas
20
21 # Basic Towers-of-Hanoi algorithm: move n pieces from a to b, using c
22 # as temporary. For each move, call report()
23 def hanoi(n, a, b, c, report):
24     if n <= 0: return
25     hanoi(n-1, a, c, b, report)
26     report(n, a, b)
27     hanoi(n-1, c, b, a, report)
28
29 # The graphical interface
30 class Tkanoi:
31     # Create our objects
32     def __init__(self, n, bitmap = None):
33         self.n = n
34         self.tk = tk = Tk()
35         self.canvas = c = Canvas(tk)
36         c.pack()
37         width, height = tk.getint(c['width']), tk.getint(c['height'])
38
39         # Add background bitmap
40         if bitmap:
41             self.bitmap = c.create_bitmap(width//2, height//2,
42                                         bitmap=bitmap,
43                                         foreground='blue')
44
45         # Generate pegs
46         pegwidth = 10
47         pegheight = height//2
48         pegdist = width//3
49         x1, y1 = (pegdist-pegwidth)//2, height*1//3
50         x2, y2 = x1+pegwidth, y1+pegheight
51         self.pegs = []
52         p = c.create_rectangle(x1, y1, x2, y2, fill='black')
53         self.pegs.append(p)
54         x1, x2 = x1+pegdist, x2+pegdist
55         p = c.create_rectangle(x1, y1, x2, y2, fill='black')
56         self.pegs.append(p)
57         x1, x2 = x1+pegdist, x2+pegdist
58         p = c.create_rectangle(x1, y1, x2, y2, fill='black')
59         self.pegs.append(p)
60         self.tk.update()
61
62         # Generate pieces
```

```

64     maxpiecewidth = pegdist*2//3
65     minpiecewidth = 2*pegwidth
66     self.pegstate = [[], [], []]
67     self.pieces = {}
68     x1, y1 = (pegdist-maxpiecewidth)//2, y2-pieceheight-2
69     x2, y2 = x1+maxpiecewidth, y1+pieceheight
70     dx = (maxpiecewidth-minpiecewidth) // (2*max(1, n-1))
71     for i in range(n, 0, -1):
72         p = c.create_rectangle(x1, y1, x2, y2, fill='red')
73         self.pieces[i] = p
74         self.pegstate[0].append(i)
75         x1, x2 = x1 + dx, x2-dx
76         y1, y2 = y1 - pieceheight-2, y2-pieceheight-2
77         self.tk.update()
78         self.tk.after(25)
79
80     # Run -- never returns
81     def run(self):
82         while 1:
83             hanoi(self.n, 0, 1, 2, self.report)
84             hanoi(self.n, 1, 2, 0, self.report)
85             hanoi(self.n, 2, 0, 1, self.report)
86             hanoi(self.n, 0, 2, 1, self.report)
87             hanoi(self.n, 2, 1, 0, self.report)
88             hanoi(self.n, 1, 0, 2, self.report)
89
90     # Reporting callback for the actual hanoi function
91     def report(self, i, a, b):
92         if self.pegstate[a][-1] != i: raise RuntimeError # Assertion
93         del self.pegstate[a][-1]
94         p = self.pieces[i]
95         c = self.canvas
96
97         # Lift the piece above peg a
98         ax1, ay1, ax2, ay2 = c.bbox(self.pegs[a])
99         while 1:
100             x1, y1, x2, y2 = c.bbox(p)
101             if y2 < ay1: break
102             c.move(p, 0, -1)
103             self.tk.update()
104
105         # Move it towards peg b
106         bx1, by1, bx2, by2 = c.bbox(self.pegs[b])
107         newcenter = (bx1+bx2)//2
108         while 1:
109             x1, y1, x2, y2 = c.bbox(p)
110             center = (x1+x2)//2
111             if center == newcenter: break
112             if center > newcenter: c.move(p, -1, 0)
113             else: c.move(p, 1, 0)
114             self.tk.update()
115
116         # Move it down on top of the previous piece
117         pieceheight = y2-y1
118         newbottom = by2 - pieceheight*len(self.pegstate[b]) - 2
119         while 1:
120             x1, y1, x2, y2 = c.bbox(p)
121             if y2 >= newbottom: break
122             c.move(p, 0, 1)
123             self.tk.update()
124
125         # Update peg state
126         self.pegstate[b].append(i)

```

```
130 import sys
131
132 # First argument is number of pegs, default 4
133 if sys.argv[1:]:
134     n = int(sys.argv[1])
135 else:
136     n = 4
137
138 # Second argument is bitmap file, default none
139 if sys.argv[2:]:
140     bitmap = sys.argv[2]
141     # Reverse meaning of leading '@' compared to Tk
142     if bitmap[0] == '@': bitmap = bitmap[1:]
143     else: bitmap = '@' + bitmap
144 else:
145     bitmap = None
146
147 # Create the graphical objects...
148 h = Tkhanoi(n, bitmap)
149
150 # ...and run!
151 h.run()
152
153
154 # Call main when run as script
155 if __name__ == '__main__':
156     main()
157
158 ###
```