```
#py
#                       MatDeck as Python Editor

# This is an example which shows how to complete a MatDeck document in Python
code. Using this way, the MatDeck Code Editor serves as the Python editor.

"""
Animated Towers of Hanoi using Tk with optional bitmap file in background.

Usage: hanoi.py [n [bitmapfile]]

n is the number of pieces to animate; default is 4, maximum 15.

The bitmap file can be any X11 bitmap file (look in /usr/include/X11/bitmaps
for
samples); it is displayed as the background of the animation.  Default is no
bitmap.
"""

from tkinter import Tk, Canvas

# Basic Towers-of-Hanoi algorithm: move n pieces from a to b, using c
# as temporary.  For each move, call report()
def hanoi(n, a, b, c, report):
    if n <= 0: return
    hanoi(n-1, a, c, b, report)
    report(n, a, b)
    hanoi(n-1, c, b, a, report)


# The graphical interface
class Tkhanoi:

    # Create our objects
    def __init__(self, n, bitmap = None):
        self.n = n
        self.tk = tk = Tk()
        self.canvas = c = Canvas(tk)
        c.pack()
        width, height = tk.getint(c['width']), tk.getint(c['height'])

        # Add background bitmap
        if bitmap:
            self.bitmap = c.create_bitmap(width//2, height//2,
                                          bitmap=bitmap,
                                          foreground='blue')

        # Generate pegs
        pegwidth = 10
        pegheight = height//2
        pegdist = width//3
        x1, y1 = (pegdist-pegwidth)//2, height*1//3
        x2, y2 = x1+pegwidth, y1+pegheight
        self.pegs = []
        p = c.create_rectangle(x1, y1, x2, y2, fill='black')
        self.pegs.append(p)
        x1, x2 = x1+pegdist, x2+pegdist
        p = c.create_rectangle(x1, y1, x2, y2, fill='black')
        self.pegs.append(p)
        x1, x2 = x1+pegdist, x2+pegdist
        p = c.create_rectangle(x1, y1, x2, y2, fill='black')
        self.pegs.append(p)
        self.tk.update()

        # Generate pieces
```

```
        maxpiecewidth = pegdist*2//3
        minpiecewidth = 2*pegwidth
        self.pegstate = [[], [], []]
        self.pieces = {}
        x1, y1 = (pegdist-maxpiecewidth)//2, y2-pieceheight-2
        x2, y2 = x1+maxpiecewidth, y1+pieceheight
        dx = (maxpiecewidth-minpiecewidth) // (2*max(1, n-1))
        for i in range(n, 0, -1):
            p = c.create_rectangle(x1, y1, x2, y2, fill='red')
            self.pieces[i] = p
            self.pegstate[0].append(i)
            x1, x2 = x1 + dx, x2-dx
            y1, y2 = y1 - pieceheight-2, y2-pieceheight-2
            self.tk.update()
            self.tk.after(25)

    # Run -- never returns
    def run(self):
        while 1:
            hanoi(self.n, 0, 1, 2, self.report)
            hanoi(self.n, 1, 2, 0, self.report)
            hanoi(self.n, 2, 0, 1, self.report)
            hanoi(self.n, 0, 2, 1, self.report)
            hanoi(self.n, 2, 1, 0, self.report)
            hanoi(self.n, 1, 0, 2, self.report)

    # Reporting callback for the actual hanoi function
    def report(self, i, a, b):
        if self.pegstate[a][-1] != i: raise RuntimeError # Assertion
        del self.pegstate[a][-1]
        p = self.pieces[i]
        c = self.canvas

        # Lift the piece above peg a
        ax1, ay1, ax2, ay2 = c.bbox(self.pegs[a])
        while 1:
            x1, y1, x2, y2 = c.bbox(p)
            if y2 < ay1: break
            c.move(p, 0, -1)
            self.tk.update()

        # Move it towards peg b
        bx1, by1, bx2, by2 = c.bbox(self.pegs[b])
        newcenter = (bx1+bx2)//2
        while 1:
            x1, y1, x2, y2 = c.bbox(p)
            center = (x1+x2)//2
            if center == newcenter: break
            if center > newcenter: c.move(p, -1, 0)
            else: c.move(p, 1, 0)
            self.tk.update()

        # Move it down on top of the previous piece
        pieceheight = y2-y1
        newbottom = by2 - pieceheight*len(self.pegstate[b]) - 2
        while 1:
            x1, y1, x2, y2 = c.bbox(p)
            if y2 >= newbottom: break
            c.move(p, 0, 1)
            self.tk.update()

        # Update peg state
        self.pegstate[b].append(i)


def main():
```

```
    import sys

    # First argument is number of pegs, default 4
    if sys.argv[1:]:
        n = int(sys.argv[1])
    else:
        n = 4

    # Second argument is bitmap file, default none
    if sys.argv[2:]:
        bitmap = sys.argv[2]
        # Reverse meaning of leading '@' compared to Tk
        if bitmap[0] == '@': bitmap = bitmap[1:]
        else: bitmap = '@' + bitmap
    else:
        bitmap = None

    # Create the graphical objects...
    h = Tkhanoi(n, bitmap)

    # ...and run!
    h.run()


# Call main when run as script
if __name__ == '__main__':
    main()

###
```