```python
#!/usr/bin/python2
### Control the LEDs in the ZooII Monarch butterfly arena.
###
### This is a library module that is meant to be imported by experiment scripts.
### It provides functions to initialize the arena, and to draw a variety of
### basic shapes that may be combined as needed. It also offers a simple
### commandline interface to control the arena lights.
###
### Daniel Vedder, April 2018 <daniel.vedder@stud-mail.uni-wuerzburg.de>
### University of Wuerzburg, Center for Computational and Theoretical Biology
### Licensed under the terms of the MIT License

import sys, math, copy
from dotstar import Adafruit_DotStar

global MODE = "SERIAL" #options: "SERIAL" / "PARALLEL"

## LED STRIP SETUP

global height, width
height = 16 #default: 16
width = 128 #default: 128

global strip
strip = Adafruit_DotStar(height*width, 2000000)
strip.begin()

global colours
colours = {"red":(strip.Color(0, 15, 0), "R"),
           "green":(strip.Color(1, 0, 0), "G"),
           "blue":(strip.Color(0, 0, 1), "B"),
           "black":(strip.Color(0, 0, 0), "-"),
           "orange":(strip.Color(2,5,0), "O"),
           "magenta":(strip.Color(0, 5, 7), "M"),
           "yellow":(strip.Color(10, 10, 0), "Y"),
           "cyan":(strip.Color(10, 0, 10), "C")}

## ARENA DEFINITIONS

global arena, old_arena
arena = ["black"] * height * width
old_arena = copy.copy(arena)

def clear_arena(colour="black", show=True):
    "Reset the arena to a given colour (default: off)"
    global arena, height, width
    arena = [colour] * height * width
    if show: draw_arena()

def pixel_id(x, y):
    "Get the LED ID of a given set of coordinates"
    global height, width
    if x < 0 or y < 0 or x >= width or y >= height:
        raise Exception(str(x)+"/"+str(y)+" is out of bounds.")
    if x%2 == 1:
        return (x+1)*height - y - 1
    else: return x*height + y

def pixel(x,y):
    "Get the colour of this pixel."
    global arena
    return arena[pixel_id(x,y)]

## DRAWING FUNCTIONS

def set_pixel(x,y,colour):
    "Set the colour of a single pixel."
    global arena
    arena[pixel_id(x,y)] = colour

def draw_arena():
    "Draw the current state of the arena to the device."
    global height, width, colour, strip, arena, old_arena
    #TODO This needs to be changed to accomodate the parallel mode
    for y in range(height):
        for x in range(width):
            colour = pixel(x,y)
            old_colour = old_arena[pixel_id(x,y)]
            if colour != old_colour:
                strip.setPixelColor(pixel_id(x,y), colours[pixel(x,y)][0])
    old_arena = copy.copy(arena)
    strip.show()

def print_arena():
    "Print out a text representation of the current state of the arena."
    global height, width, colours
    for y in range(height):
        for x in range(width):
            sys.stdout.write(colours[pixel(x,y)][1])
        sys.stdout.write('\n')
        sys.stdout.flush()

def draw_shape(coords, colour="green", flush=True, draw=True):
    '''
    Draw a shape from a list of coordinates (as produced by the shape functions).
    colour: The colour to use
    flush: If true, will output the result immediately
    draw: If true and flush is true, output to device, otherwise print to screen
    '''
    for c in coords:
        set_pixel(c[0], c[1], colour)
    if flush:
        if draw: draw_arena()
        else: print_arena()

## SHAPE DEFINITIONS
## A shape is a list of coordinate tuples whose pixels are to be drawn

def line(x1, y1, x2, y2):
    "A straight line from x1/y1 to x2/y2"
    if x1 == x2:
        return vertical_line(x1, y1, y2)
    elif y1 == y2:
        return horizontal_line(y1, x1, x2)
    else: return diagonal_line(x1, y1, x2, y2)

def horizontal_line(y, x1, x2):
    shape = []
    if x2 < x1: x1,x2 = x2,x1
    for x in range(x1, x2+1):
        shape.append((x, y))
    return shape

def vertical_line(x, y1, y2):
    shape = []
    if y2 < y1: y1,y2 = y2,y1
    for y in range(y1, y2+1):
        shape.append((x, y))
    return shape

def diagonal_line(x1, y1, x2, y2):
    if x2 < x1:
        x1,x2 = x2, x1
        y1,y2 = y2, y1
    shape = []
    slope = (x2-x1)/(y2-y1)
    # XXX a bit ugly, but it works
    if abs(slope) < 1: # steep lines
        for y in range(y1, y2+1):
            x = int(round(x2 - ((y2-y)*slope)))
            shape.append((x,y))
    else: # shallow lines
        for x in range(x1, x2+1):
            y = int(round(y2 - ((x2-x)/slope)))
            shape.append((x,y))
    return shape

def polygon(corners):
    "A polygon connecting each set of coordinates passed to it via straight lines"
    shape = line(corners[len(corners)-1][0], corners[len(corners)-1][1],
```

```python
                  corners[0][0], corners[0][1])
    for c in range(len(corners)-1):
        shape.extend(line(corners[c][0], corners[c][1],
                          corners[c+1][0], corners[c+1][1]))
    return shape

def triangle(x1, y1, x2, y2, x3, y3):
    corners = ((x1,y1), (x2,y2), (x3,y3))
    return polygon(corners)

def rectangle(x1, y1, x2, y2, x3, y3, x4, y4, filled=False):
    '''
    Draw a shape with four sides (doesn't strictly have to be a rectangle).
    filled: if false, simply returns the outline
    '''
    corners = ((x1,y1), (x2,y2), (x3,y3), (x4,y4))
    outline = polygon(corners)
    if not filled: return outline
    shape = outline
    for x in range(min(x1,x2,x3,x4), max(x1,x2,x3,x4)+1):
        for y in range(min(y1,y2,y3,y4), max(y1,y2,y3,y4)+1):
            conds = []
            for c in outline:
                # Every point inside the rectangle has, on the same
                # axis, one point larger and one smaller than itself
                if x == c[0] and y < c[1]: conds.append("yl")
                elif x == c[0] and y > c[1]: conds.append("yg")
                if y == c[1] and x < c[0]: conds.append("xl")
                elif y == c[1] and x > c[0]: conds.append("xg")
            if "yl" in conds and "yg" in conds and "xl" in conds and "xg" in conds:
                shape.append((x,y))
    return shape

def circle(center_x, center_y, radius, filled=False, quarters=[1,2,3,4]):
    '''
    Draw a circle, defined by its center point and radius.
    filled: if false, will only draw the outline
    quarters: quarters of the circle to draw (1.TR, 2.BR, 3.BL, 4.TL)
    '''
    shape = []
    for x in range(radius+1):
        for y in range(radius+1):
            distance = round(math.sqrt(x**2+y**2))
            if distance == radius or (filled and distance <= radius):
                if 1 in quarters: shape.append((center_x+x, center_y-y))
                if 2 in quarters: shape.append((center_x+x, center_y+y))
                if 3 in quarters: shape.append((center_x-y, center_y+x))
                if 4 in quarters: shape.append((center_x-y, center_y-x))
    return shape

## COMMANDLINE INTERFACE

def parseArgs():
    '''
    A rudimentary commandline interface. Usage:
    ./arena.py clear [colour]
    ./arena.py set <x> <y> <colour>
    Use `--serial` or `--parallel` before the clear/set command to choose a mode.
    '''
    global MODE, colours
    #XXX Is it sensible to set the mode via commandline?
    if "--serial" in sys.argv:
        MODE = "SERIAL"
    elif "--parallel" in sys.argv:
        MODE = "PARALLEL"
    elif "clear" in sys.argv:
        if sys.argv[-1] in colours.keys():
            clear_arena(sys.argv[-1])
        else: clear_arena()
    elif "set" in sys.argv:
        x = sys.argv[-3]
        y = sys.argv[-2]
        c = sys.argv[-1]
        if x.isdigit() and y.isdigit() and c in colours.keys():
            set_pixel(int(x),int(y),c)
        else:
            print "Usage: ./arena.py set <x> <y> <colour>"
            return
    draw_arena()

if __name__ == '__main__':
    parseArgs()
```