"""
Soft Des Spring 2020 Mini Project 2: Computational Art
@author: Sam Coleman
"""

import random
from PIL import Image
from math import pi, sin, cos

def build_random_function(min_depth, max_depth):
"""Build a random function.

```
Builds a random function of depth at least min_depth and depth at most
max_depth. (See the assignment write-up for the definition of depth
in this context)

Args:
    min_depth: the minimum depth of the random function
    max_depth: the maximum depth of the random function

Returns:
    The randomly generated function represented as a nested list.
    (See the assignment write-up for details on the representation of
    these functions)
"""
# TODO: implement this
function = []
operations = ['x', 'y', 'prod', 'avg', 'cos_pi', 'sin_pi']
random_func = random.randint(2,5)
random_var = random.randint(0,1)
end_possibility = random.randint(0,2)

function.append(operations[random_func])

if min_depth < 2:
    return function.append(operations[random_var])
else:
    return build_random_function(min_depth - 1, max_depth - 1)
```

build_random_function(7,9)

def evaluate_random_function(f, x, y):
"""Evaluate the random function f with inputs x,y.

```
The representation of the function f is defined in the assignment write-up.
```

```
    Args:
        f: the function to evaluate
        x: the value of x to be used to evaluate the function
        y: the value of y to be used to evaluate the function

    Returns:
        The function value

    Examples:
        >>> evaluate_random_function(["x"],-0.5, 0.75)
        -0.5
        >>> evaluate_random_function(["y"],0.1,0.02)
        0.02
    """
    # TODO: implement this
    if f[0] =="x":
        return x
    elif f[0] == "y":
        return y
    elif f[0] == "prod":
        return evaluate_random_function(f[1], x, y) * evaluate_random_function(f[2], x,
    elif f[0] == "avg":
        return .5 * (evaluate_random_function(f[1], x, y) + evaluate_random_function(f[
    elif f[0] == "cos_pi":
        return cos(pi * evaluate_random_function(f[1], x, y))
    elif f[0] == "sin_pi":
        return sin(pi * evaluate_random_function(f[1], x, y))
```

def remap_interval(val,
input_interval_start,
input_interval_end,
output_interval_start,
output_interval_end):
"""Remap a value from one interval to another.

```
    Given an input value in the interval [input_interval_start,
    input_interval_end], return an output value scaled to fall within
    the output interval [output_interval_start, output_interval_end].

    Args:
        val: the value to remap
        input_interval_start: the start of the interval that contains all
                              possible values for val
        input_interval_end: the end of the interval that contains all possible
                            values for val
        output_interval_start: the start of the interval that contains all
                               possible output values
        output_inteval_end: the end of the interval that contains all possible
                            output values
```

```
    Returns:
        The value remapped from the input to the output interval

    Examples:
        >>> remap_interval(0.5, 0, 1, 0, 10)
        5.0
        >>> remap_interval(5, 4, 6, 0, 2)
        1.0
        >>> remap_interval(5, 4, 6, 1, 2)
        1.5
    """
    # TODO: implement this

    re_mapped = output_interval_start + (output_interval_end - output_interval_start) *
    return re_mapped
```

def color_map(val):
"""Maps input value between -1 and 1 to an integer 0-255, suitable for use as an RGB color code.

```
    Args:
        val: value to remap, must be a float in the interval [-1, 1]

    Returns:
        An integer in the interval [0,255]

    Examples:
        >>> color_map(-1.0)
        0
        >>> color_map(1.0)
        255
        >>> color_map(0.0)
        127
        >>> color_map(0.5)
        191
    """
    # NOTE: This relies on remap_interval, which you must provide
    color_code = remap_interval(val, -1, 1, 0, 255)
    return int(color_code)
```

def test_image(filename, x_size=350, y_size=350):
"""Generate a test image with random pixels and save as an image file.

```
    Args:
        filename: string filename for image (should be .png)
        x_size, y_size: optional args to set image dimensions (default: 350)
    """
    # Create image and loop over all pixels
```

```
    im = Image.new("RGB", (x_size, y_size))
    pixels = im.load()
    for i in range(x_size):
        for j in range(y_size):
            x = remap_interval(i, 0, x_size, -1, 1)
            y = remap_interval(j, 0, y_size, -1, 1)
            pixels[i, j] = (random.randint(0, 255),  # Red channel
                            random.randint(0, 255),  # Green channel
                            random.randint(0, 255))  # Blue channel

    im.save(filename)
```

def generate_art(filename, x_size=350, y_size=350):
"""Generate computational art and save as an image file.

```
    Args:
        filename: string filename for image (should be .png)
        x_size, y_size: optional args to set image dimensions (default: 350)
    """
    # Functions for red, green, and blue channels - where the magic happens!
    red_function = ["x"]
    green_function = ["y"]
    blue_function = ["x"]
    #red_function = build_random_function(7, 9)
    #green_function = build_random_function(7, 9)
    #blue_function = build_random_function(7, 9)
    # Create image and loop over all pixels
    im = Image.new("RGB", (x_size, y_size))
    pixels = im.load()
    for i in range(x_size):
        for j in range(y_size):
            x = remap_interval(i, 0, x_size, -1, 1)
            y = remap_interval(j, 0, y_size, -1, 1)
            pixels[i, j] = (
                color_map(evaluate_random_function(red_function, x, y)),
                color_map(evaluate_random_function(green_function, x, y)),
                color_map(evaluate_random_function(blue_function, x, y))
            )

    im.save(filename)
```

if **name** == '**main**':
import doctest
#doctest.testmod()
doctest.run_docstring_examples(remap_interval, globals(), verbose = False)

```
# Create some computational art!
# TODO: Un-comment the generate_art function call after you
```

```
#       implement remap_interval and evaluate_random_function
#generate_art("myart1.png")

# Test that PIL is installed correctly
# TODO: Comment or remove this function call after testing PIL install
#test_image("noise.png")
```