Graham Collins

Completed



Originals





```
from mediaComp import *

# Name: Graham Collins
# Date: 10/24/2026

# Main function that builds the entire photo collage
def collage():
    # Load all image files from the project folder
    picture_1 = makePicture(getMediaFolder("sky.jpg"))
    picture_2 = makePicture(getMediaFolder("sarah.jpg"))
    signature = makePicture(getMediaFolder("signature.png"))

    # Setup the starting canvas
    canvas = makeEmptyPicture(736, 736)
    w = getWidth(canvas)
    h = getHeight(canvas)

    # Background image merge
    canvas = merge(picture_1, picture_2, canvas)

    # Sky Inset (1/3 of the way down)
    canvas = sky(picture_1, canvas, start_offset = 75, insert_size = 175)

    # Sarah Inset (2/3 of the way down)
    canvas = sarah(picture_2, canvas, start_offset = 280, insert_size = 175)

    # Top Left corner (mirrored)
    canvas = draw_corner(picture_2, canvas, start_x = 25, start_y = 25, insert_size = 50, mirror = True)

    # Top right corner (pink)
    canvas = draw_corner(picture_2, canvas, start_x = w - 25 - 50, start_y = 25, insert_size = 50, pink = True)

    # Bottom left corner (mirrored and green)
    canvas = draw_corner(picture_2, canvas, start_x=25, start_y=h-25-50, insert_size=50, mirror=True, green=True)
```

```
    # Bottom right corner
    canvas = draw_corner(picture_2, canvas, start_x = w - 25 - 50, start_y = h - 25 - 50, insert_size = 50)

    # Draws the repeating border around the edge (Runs before the signaute layer to avoid overwritng it)
    canvas = draw_border(picture_2, canvas, tile_size = 25)

    # Apply Signature (Scales cleanly across layout)
    final_image = apply_signature(signature, canvas)

    # Displays completed collage image
    pictureTool(final_image)

# Combines picture 1 and picture 2 across the canvas
def merge(picture_1, picture_2, canvas):
    canvas = copy(picture_2, canvas, start = 0) # Copy onto even lines
    canvas = copy(picture_1, canvas, start = 1) # Copy onto odd lines
    return canvas

# Loops through every other column to interalce two images together for background
def copy(source_pic, dest_canvas, start):
    dest_w = getWidth(dest_canvas)
    dest_h = getHeight(dest_canvas)
    source_w = getWidth(source_pic)
    source_h = getHeight(source_pic)
    for dest_x in range(start, dest_w, 2):
        for dest_y in range(0, dest_h, 1):
            apply_pixel(source_pic, dest_canvas, dest_x, dest_y, 0, 0, dest_w, dest_h,
                        mirror = False, pink = False, green = False)
    return dest_canvas

# Creates the center sky window box with a white border
def sky(source_picture, dest_canvas,start_offset, insert_size):
    dest_h = getHeight(dest_canvas)
    target_start_x = start_offset
    target_start_y = int(dest_h / 3) # Positions it at the 1/3 height goal
    border_thickness = 3

    for dest_x in range(target_start_x, target_start_x + insert_size):
        for dest_y in range(target_start_y, target_start_y + insert_size):
            # Check if we are drawing the inside or outer outline frame
            is_border_x = (dest_x < target_start_x + border_thickness) or
                          (dest_x >= target_start_x + insert_size - border_thickness)
            is_border_y = (dest_y < target_start_y + border_thickness) or
                          (dest_y >= target_start_y + insert_size - border_thickness)
```

```
            if is_border_x or is_border_y:
                dest_pixel = getPixelAt(dest_canvas, dest_x, dest_y)
                setColor(dest_pixel, makeColor(255, 255, 255)) # Draws the white border frame lines
            else:
                apply_pixel(source_picture, dest_canvas, dest_x, dest_y, target_start_x, target_start_y,
                                insert_size, insert_size, mirror = False, pink = False, green = False)
    return dest_canvas

# Creates the middle Sarah window box with a white border
def sarah(source_picture, dest_canvas, start_offset, insert_size):
    dest_h = getHeight(dest_canvas)
    target_start_x = start_offset
    target_start_y = int(dest_h * (2.0 / 3.0)) # Sets it exactly 2/3 of the way down the screen
    border_thickness = 3

    for dest_x in range(target_start_x, target_start_x + insert_size):
        for dest_y in range(target_start_y, target_start_y + insert_size):
            is_border_x = (dest_x < target_start_x + border_thickness) or
                            (dest_x >= target_start_x + insert_size - border_thickness)
            is_border_y = (dest_y < target_start_y + border_thickness) or
                            (dest_y >= target_start_y + insert_size - border_thickness)
            if is_border_x or is_border_y:
                dest_pixel = getPixelAt(dest_canvas, dest_x, dest_y)
                setColor(dest_pixel, makeColor(255, 255, 255))
            else:
                apply_pixel(source_picture, dest_canvas, dest_x, dest_y, target_start_x, target_start_y,
                            insert_size, insert_size, mirror = False, pink = False, green = False)
    return dest_canvas

# Reverse chromakey function to get signature onto canvas
def apply_signature(sig_pic, dest_canvas):
    sig_w = getWidth(sig_pic)
    sig_h = getHeight(sig_pic)
    canvas_w = getWidth(dest_canvas)
    canvas_h = getHeight(dest_canvas)

    for dest_x in range(0, canvas_w):
        for dest_y in range(0, canvas_h):
            # Coordinates are automatically scaled to handle different canvas sizes cleanly
            source_x = calc_scale(dest_x, canvas_w, sig_w)
            source_y = calc_scale(dest_y, canvas_h, sig_h)

            sig_pixel = getPixelAt(sig_pic, source_x, source_y)
            r = getRed(sig_pixel)
```

```
            g = getGreen(sig_pixel)
            b = getBlue(sig_pixel)

            # Reverse chromakey check: ignores the background white space and draw the pen lines
            if not (r >= 200 and g >= 200 and b >= 200):
                dest_pixel = getPixelAt(dest_canvas, dest_x, dest_y)
                setColor(dest_pixel, getColor(sig_pixel))
    return dest_canvas

# Corner tool that accepts flags for filtering and mirroring
def draw_corner(source_pic, dest_canvas, start_x, start_y, insert_size,
                mirror = False, pink = False, green = False):
    for dest_x in range(start_x, start_x + insert_size):
        for dest_y in range(start_y, start_y + insert_size):
            apply_pixel(source_pic, dest_canvas, dest_x, dest_y, start_x, start_y, insert_size, insert_size,
                        mirror, pink, green)
    return dest_canvas

# Main loop for mapping the tiled border pattern around the outer borders of the canvas
def draw_border(border_pic, dest_canvas, tile_size):
    dest_w = getWidth(dest_canvas)
    dest_h = getHeight(dest_canvas)
    for dest_x in range(0, dest_w):
        for dest_y in range(0, dest_h):
            # Check if current coordinates hit the border zones
            top = dest_y < tile_size
            bottom = dest_y >= (dest_h - tile_size)
            left = dest_x < tile_size
            right = dest_x >= (dest_w - tile_size)

            if top or bottom or left or right:
                apply_border_pixel(border_pic, dest_canvas, dest_x, dest_y, tile_size)
    return dest_canvas

# Figures out sizing ranges and copies the pixel values over to the destination canvas
def apply_pixel(source_pic, dest_canvas, dest_x, dest_y, target_start_x, target_start_y, target_w, target_h,
                mirror, pink, green):
    source_x = calc_scale(dest_x - target_start_x, target_w, getWidth(source_pic))
    source_y = calc_scale(dest_y - target_start_y, target_h, getHeight(source_pic))

    # Flip the x position if mirror transformation is enabled
    if mirror:
        source_x = calc_mirror(source_x, getWidth(source_pic))
    copy_pixel_color(source_pic, dest_canvas, source_x, source_y, dest_x, dest_y, pink, green)
```

```
# Handles specific coordinate mapping paths for the border loops
def apply_border_pixel(source_pic, dest_canvas, dest_x, dest_y, tile_size):
    local_x = calc_border_track(dest_x, getWidth(dest_canvas), tile_size)
    local_y = calc_border_track(dest_y, getHeight(dest_canvas), tile_size)

    source_x = calc_scale(local_x, tile_size, getWidth(source_pic))
    source_y = calc_scale(local_y, tile_size, getHeight(source_pic))
    copy_pixel_color(source_pic, dest_canvas, source_x, source_y, dest_x, dest_y, pink = False, green = False)

# Aligns right and bottom image margins back down to index 0 so pattern remains clean
def calc_border_track(current_pos, max_limit, tile_size):
    if current_pos < tile_size:
        return current_pos
    elif current_pos >= (max_limit - tile_size):
        return current_pos - (max_limit - tile_size)
    return current_pos % tile_size

# Scales source image pixel coordinates to target sizing boundaries
def calc_scale(current_pos, total_dest_range, total_source_range):
    scaled_pos = int((current_pos / total_dest_range) * total_source_range)
    # Caps the index range to prevent out of bounds crashes
    if scaled_pos >= total_source_range:
        return total_source_range - 1
    if scaled_pos < 0:
        return 0
    return scaled_pos

# Flips horizontal alignment around a midpoint line
def calc_mirror(current_x, max_width):
    return (max_width - 1) - current_x

# Color modification: Shifts color channels toward a pink tint
def calc_pink(source_pixel):
    r = getRed(source_pixel)
    g = getGreen(source_pixel)
    b = getBlue(source_pixel)
    pink_r = min(int(r * 1.2), 255) # Keeps color value safely capped at 255 max
    pink_g = int(g * 0.4)
    pink_b = min(int(b * 1.2), 255)
    return makeColor(pink_r, pink_g, pink_b)

# Color modification: Shifts color channels toward a green tint
def calc_green(source_pixel):
    r = getRed(source_pixel)
    g = getGreen(source_pixel)
```

```
    b = getBlue(source_pixel)
    green_r = int(r * 0.4)
    green_g = min(int(g * 1.2), 255)
    green_b = int(b * 0.4)
    return makeColor(green_r, green_g, green_b)

# Checks pixel and writes colors directly to the destination coordinates
def copy_pixel_color(source_pic, dest_canvas, src_x, src_y, dest_x, dest_y, pink, green):
    source_pixel = getPixelAt(source_pic, src_x, src_y)
    if pink:
        color = calc_pink(source_pixel)
    elif green:
        color = calc_green(source_pixel)
    else:
        color = getColor(source_pixel)
    dest_pixel = getPixelAt(dest_canvas, dest_x, dest_y)
    setColor(dest_pixel, color)
```