🖼️ Question 04: Image Augmentation Pipeline

Goal: Build a flexible "pipeline" to apply a series of transformations (like flip and adjust brightness) to simulate image data augmentation for an AI training team. **Topics:** OOP (Classes, Objects), Functions as Arguments, `lambda`, List Comprehensions.

Provided Data

A sample 2x3 image represented as a 2D list:

In [6]:
```
originalPixels = [
    [10, 20, 30],
    [40, 50, 60]
]
```

Tasks

1. **Image Class:**

   - Create an **`Image` class** that stores a 2D list of pixels.
   - It must have an `applyTransformation(transformationFunc)` method that accepts a function as an argument and uses it to update its own pixel data.

1. **Manual Deep Copy:**

   - Transformations must **not change the original image**.
   - Implement a `getCopy()` method for your `Image` class **without using the `copy` module**.
   - *Hint: A list comprehension can create a new, independent 2D list.*

In [7]:
```
class Image:
    def __init__(self, pixels):
        self.pixels = pixels

    def applyTransformation(self, transformationFunc):
        # Assuming transformationFunc takes the pixel data and retur
ns new pixel data
        self.pixels = transformationFunc(self.pixels)

    def getCopy(self):
        # Manual deep copy without using the copy module
        new_pixels = [row[:] for row in self.pixels]
        return Image(new_pixels)
```

1. **Transformation Functions:** Write standalone functions that each take a `pixelData` list and return a **new 2D list**:

   - `flipHorizontal(pixelData)`
   - `adjustBrightness(pixelData, brightnessValue)` : This one needs an extra argument.
   - `rotateNinetyDegrees(pixelData)`

In [8]:
```
def flipHorizontal(pixelData):
    # Create a new list of lists by reversing each row
    return [row[::-1] for row in pixelData]

def adjustBrightness(pixelData, brightnessValue):
    # Create a new list of lists with adjusted pixel values
    # Ensure pixel values stay within a valid range (e.g., 0-255)
    return [[max(0, min(255, pixel + brightnessValue)) for pixel in
row] for row in pixelData]

def rotateNinetyDegrees(pixelData):
    # Get the number of rows and columns
    rows = len(pixelData)
    cols = len(pixelData[0]) if rows > 0 else 0

    # Create a new list of lists for the rotated image
    # The new image will have dimensions cols x rows
    rotated_pixels = [[0 for _ in range(rows)] for _ in range(cols)]

    # Populate the rotated image
    for r in range(rows):
        for c in range(cols):
            rotated_pixels[c][rows - 1 - r] = pixelData[r][c]

    return rotated_pixels
```

1. **Pipeline Class:**

   - Create an **`AugmentationPipeline` class**.
   - It should have an `addStep(transformFunc)` method to store a list of functions.
   - Its `processImage(originalImage)` method should loop through this list, apply each function to a **fresh copy** of the `originalImage` , and return a **new list of all the augmented images**.

In [9]:
```
class AugmentationPipeline:
    def __init__(self):
        self.transformations = []

    def addStep(self, transformFunc):
        self.transformations.append(transformFunc)

    def processImage(self, originalImage):
        augmented_images = []
        for transformFunc in self.transformations:
            # Get a fresh copy of the original image for each transf
ormation
            current_image = originalImage.getCopy()
            # Apply the transformation
            current_image.applyTransformation(transformFunc)
            # Add the augmented image to the list
            augmented_images.append(current_image)
        return augmented_images
```

1. **Using lambda:**

- The `adjustBrightness` function needs two arguments, but the pipeline's `applyTransformation` method only passes one.
- When adding this step to your pipeline, use a **`lambda` function to wrap the `adjustBrightness` call**, "pre-loading" it with the `brightnessValue` you want.

In [10]:
```
# Create an instance of the Image class
original_image = Image(originalPixels)

# Create an instance of the AugmentationPipeline class
pipeline = AugmentationPipeline()

# Add transformations to the pipeline
pipeline.addStep(flipHorizontal)
pipeline.addStep(rotateNinetyDegrees)

# Add adjustBrightness using a lambda function to specify the bright
ness value
brightness_value = 50  # Example brightness value
pipeline.addStep(lambda pixels: adjustBrightness(pixels, brightness_
value))

# Process the original image through the pipeline
augmented_images = pipeline.processImage(original_image)

# Display the original and augmented images
print("Original Image:")
print(original_image.pixels)

print("\nAugmented Images:")
for i, img in enumerate(augmented_images):
    print(f"Transformation {i+1}:")
    print(img.pixels)
```

```
Original Image:
[[10, 20, 30], [40, 50, 60]]

Augmented Images:
Transformation 1:
[[30, 20, 10], [60, 50, 40]]
Transformation 2:
[[40, 10], [50, 20], [60, 30]]
Transformation 3:
[[60, 70, 80], [90, 100, 110]]
```