Learning Objectives

Learners will be able to...

- **Understand the concept of image transformations and their significance in image processing.**
- **Differentiate between spatial transformations and color transformations.**
- **Apply various spatial transformations, such as flipping, rotating, and inverting images, using the Python Imaging Library (PIL)**.
- **Use the DALL-E 2 API in combination with the PIL library to create unique and interesting variations of generated images**.
- **Experiment with different transformations and combinations to create custom images and effects using the DALL-E 2 API and the PIL library.**

info

Make Sure You Know

You are familiar with Python.

Limitations

This is a gentle introduction. So there is a little bit of Python programming. The information might not be the most up to date as OpenAI releases new features.

Transformation

In the context of image processing, "**transformations**" refer to operations that modify the spatial, color, or other properties of an image. These operations can be applied to the image to enhance its appearance, correct its orientation, change its perspective, or create various artistic effects. Transformations can be broadly categorized into two types: color transformations and spatial transformations.

Color Transformations:

Color transformations involve modifying the color properties of an image, such as its brightness, contrast, saturation, or hue. Common color transformations include:

- **Inverting**: Reversing the color values of the image, creating a "negative" of the original image. This can be used to create interesting visual effects or emphasize specific features of an image.
- **Adjusting brightness**: Changing the overall intensity of the image's colors. This can be used to correct underexposed or overexposed images or create specific moods.
- **Adjusting contrast**: Modifying the difference between the darkest and lightest colors in the image. This can be used to enhance the details or create specific effects.
- **Adjusting saturation**: Changing the intensity of the colors in the image. This can be used to make the colors more vibrant or muted, or to create monochromatic or grayscale images.
- **Adjusting hue**: Shifting the colors in the image along the color spectrum. This can be used to change the overall color scheme of the image or create specific effects.

Spatial Transformations:
Spatial transformations involve changing the geometric properties of an image, such as its position, orientation, or size. Common spatial transformations include:
- **Flipping**: Mirroring the image either horizontally or vertically. This is useful for creating reflections or correcting image orientation.
- **Rotating**: Changing the orientation of the image by a specified angle. This can be used to correct image orientation or create dynamic compositions.
- **Scaling**: Changing the size of the image, either by enlarging or reducing it. This can be used to fit the image into a specific size or resolution.
- **Cropping**: Removing parts of the image outside a specified region. This can be used to focus on a specific area of the image or remove unwanted portions.
- **Warping**: Changing the perspective or shape of the image by applying a geometric transformation. This can be used to correct perspective distortion or create artistic effects.

These transformations can be applied individually or in combination to create a wide array of unique and visually engaging images. By using various image processing libraries like PIL, you can easily apply these transformations to your images and create custom effects tailored to your needs.
We have more or less covered the color transformations in the previous lesson we will be focused on spacial transformation in this one.

Saving our Image

Image Generation

In order to not have to keep copying and pasting our result in, we are going to save the image we are going to interact with as a file in our Codio box.

The first thing we are going to do is get our base image. Previously we used very simple prompts to generate simple images. The simple images were very useful to see the color transformations. Now that we are going to work with more spacial transformations we can make some cooler pictures and still easily visualize the changes. Let's start with getting our library and image generation function

```
import os
import openai
import secret
from PIL import Image,ImageOps
from io import BytesIO
import requests

openai.api_key = secret.api_key

# Generate the base image
def generate_base_image(prompt):
    response = openai.Image.create(
        prompt=prompt,
        n=1,
        size="512x512"
    )
    return response['data'][0]['url']
```

From there you can use any prompt you would like. We are going to use the following prompt:

```
cool_prompt= "Reunion of man, team, squad with katanas,
        cyberpunk, abstract, full hd render + 3d octane render
        +4k UHD + immense detail + dramatic lighting + well lit
        + black, purple, blue, pink, cerulean, teal, metallic
        colours, + fine details + octane render + 8k"

image_url = generate_base_image(cool_prompt)

img_data = requests.get(image_url).content
with open('base_img.jpg', 'wb') as handler:
    handler.write(img_data)
```

In our case the following prompt generated this pretty cool image.

If you want feel free to generate other images until you have something you like. **If** you want to use the picture above. You can run the following to get the image above. Additional images will be provided below, if you want them simply switch the corresponding source file. Example, `base_img9.jpg` or `base_img8.jpg` ...

```
import shutil

# Set the source and destination file paths
src_file = "base_Img0.jpg"
dst_file = "base_img.jpg"

# Copy the file to the new destination
shutil.copy(src_file, dst_file)
```

Comment out the code so you are no longer generating and moving your files. More choices are available including their prompts below:

img9

```
cool_prompt= "cybernetic ninja cyborg with glowing parts , ninja
        has sword, impressive, surreal, cinematic lighting,
        cinematic photoshot, extremely detailed and complex, VFX
        volume fog around, nighttime, super max, surreal, super
        detailed, high contrast, Rtx on, Hdr, photography,
        realistic, dof on, fov on, motion blur, lens flares on,
        50mm Prime f/1.8, White balance, Super resolution,
        Megapixel, ProPhoto RGB, VR, high, epic, Rear half
        lighting, Lights background, natural lighting,
        incandescent light, fiber optic, mood lighting, cinema
        lighting, studio lighting, soft illumination,
        volumetric, contrast, dark lighting, accent lighting,
        projection global illumination, Screen space global
        illumination, Ray tracing global illumination, Red
        fringing light, 45% cold color grading, Optics,
        Scattering, Glow, Shadows, hyperrealism, Caustic water,
        refraction water, exquisite detail, intricately-
        detailed, ultra-detailed photography, high-sharpness,
        high reflection, award-winning photograph"
```

base_img9

img8

```
cool_prompt= " masked mutant with  fire powers,  standing atop a
        skyscraper during a thunderstorm. body is crackling with
        electricity, with glowing veins and eyes. cyberpunk,
        abstract, full hd render + 3d octane render +4k UHD +
        immense detail + dramatic lighting + well lit + black,
        purple, blue, pink, cerulean, teal, metallic colours +
        fine details + octane render + 8k"
```

base_img8

img7

```
cool_prompt="Create an image of a futuristic city with towering
skyscrapers, advanced transportation systems, and renewable
energy sources. Use high-tech materials that shimmer and reflect
light, and incorporate green spaces and parks into the skyline.
Use dynamic and colorful lighting with neon lights and
holographic projections. Show a diverse and cosmopolitan
population with cutting-edge fashion and technology. Capture the
city's sleek and sophisticated aesthetic in your image."
```

base_img7

Flipping

Flipping an image involves mirroring it either horizontally or vertically. This can be useful for creating reflections, correcting image orientation, or generating additional variations of a generated image. we will explore horizontal and vertical flipping using the Python Imaging Library (PIL) and the DALL-E 2 API.

Horizontal Flipping

Horizontal flipping mirrors the image along its vertical axis, effectively reversing its left and right sides. This can be used to create a reflection effect or to correct an image's orientation if it appears reversed. Now feel free to comment out the code from the previous page, remember to keep your libraries. After copy and paste the following code.

```
#Saving our base image
base_image=Image.open('base_img.jpg')
# Flip the base image horizontally
flipped_horizontal_image = ImageOps.mirror(base_image)

# Save the horizontally flipped image to a file
flipped_horizontal_image.save('flipped_horizontal.jpg')
```

The base image is provided in that same lower left panel so it's easy to compare.

Vertical Flipping

Vertical flipping mirrors the image along its horizontal axis, effectively reversing its top and bottom sides. This can be useful for creating a reflection effect on a horizontal surface or correcting an image's orientation if it appears upside-down.

```
# Flip the base image vertically
flipped_vertical_image = ImageOps.flip(base_image)

# Save the vertically flipped image to a file
flipped_vertical_image.save('flipped_vertical.jpg')
```

Combining Flips

You can also combine horizontal and vertical flips to create a diagonal reflection effect or generate additional variations of the image.

```
# Flip the base image horizontally and vertically
flipped_both_image = ImageOps.mirror(ImageOps.flip(base_image))

# Save the horizontally and vertically flipped image to a file
flipped_both_image.save('flipped_both.jpg')
```

Rotating

Rotating an image involves changing its orientation by a specified angle. This can be useful for correcting image orientation, creating dynamic compositions, or generating additional variations of a generated image. In this extended example, we will explore various ways to rotate images using the Python Imaging Library (PIL) and the DALL-E 2 API. To avoid clutter we are going to use a new code file. Since we know where our base image is located we just need our libraries.

```
import os
import openai
import secret
from PIL import Image,ImageOps
from io import BytesIO
import requests

openai.api_key = secret.api_key

#Saving our base image
base_image=Image.open('base_img.jpg')
```

Basic Rotation

In its simplest form, image rotation involves specifying an angle by which to rotate the image. The image is rotated around its center by default.

```
# Rotate the base image by 45 degrees
rotated_image_45 = base_image.rotate(45)

# Save the rotated image to a file
rotated_image_45.save('rotated_45_.jpg')
```

Rotation with Resampling

When rotating an image, the original pixel values may not align with the new pixel grid. To address this issue, you can use resampling techniques, which interpolate the pixel values to create a smoother appearance. The most common resampling technique is bilinear interpolation.

```
# Rotate the base image by 45 degrees with bilinear
        interpolation
rotated_image_45_bilinear = base_image.rotate(45,
        resample=Image.BILINEAR)

# Save the rotated image with bilinear interpolation to a file
rotated_image_45_bilinear.save('rotated_45_bilinear.jpg')
```

Rotation with Custom Center

By default, the image is rotated around its center. However, you can specify a custom center for the rotation if you want to create a different effect or composition

```
# Define custom center coordinates (x, y)
center_x = 100
center_y = 100

# Rotate the base image by 45 degrees around the custom center
rotated_image_custom_center = base_image.rotate(45, center=
        (center_x, center_y))

# Save the rotated image with custom center to a file
rotated_image_custom_center.save('rotated_custom_center.jpg')
```

Rotation with Expand Option

When rotating an image, the corners may be cropped if they extend beyond the original image boundaries. To avoid this, you can use the expand option to automatically resize the image canvas to accommodate the rotated image.

```
# Rotate the base image by 120 degrees and expand the canvas
rotated_image_expanded = base_image.rotate(120, expand=True)

# Save the rotated image with expanded canvas to a file
rotated_image_expanded.save('rotated_expanded.jpg')
```

By applying various rotation techniques to images generated by the DALL-E 2 API, you can create interesting and dynamic compositions, adding visual appeal and diversity to your images.

Inverting Images

Inverting an image involves reversing its color values, creating a “negative” of the original image. This can be useful for creating interesting visual effects, emphasizing specific features of an image, or generating additional variations of a generated image. We will explore image inversion using the Python Imaging Library (PIL) and the DALL-E 2 API.

```
import os
import openai
import secret
from PIL import Image,ImageOps
from io import BytesIO
import requests

openai.api_key = secret.api_key

#Saving our base image
base_image=Image.open('base_img.jpg')
```

Color Inversion

Color inversion involves reversing the RGB values of each pixel in the image. This can create a striking effect that highlights different aspects of the image compared to the original.

```
# Invert the colors of the base image
inverted_image = ImageOps.invert(base_image)

# Save the inverted image to a file
inverted_image.save('inverted.jpg')
```

Grayscale Inversion

Grayscale inversion involves first converting the image to grayscale and then reversing its color values. This creates a negative effect while preserving the tonal relationships in the image.

```
# Convert the base image to grayscale
grayscale_image = ImageOps.grayscale(base_image)

# Invert the grayscale image
inverted_grayscale_image = ImageOps.invert(grayscale_image)

# Save the inverted grayscale image to a file
inverted_grayscale_image.save('inverted_grayscale.jpg')
```

Inversion with Custom Color Channels

You can also create custom effects by inverting only specific color channels of the image. For example, you can invert the red and green channels while preserving the blue channel.

```
# Split the color channels of the base image
r, g, b = base_image.split()

# Invert the red and green channels
inverted_r = ImageOps.invert(r)
inverted_g = ImageOps.invert(g)

# Combine the inverted red and green channels with the original
        blue channel
inverted_custom_channels_image = Image.merge("RGB", (inverted_r,
        inverted_g, b))

# Save the custom channel inverted image to a file
inverted_custom_channels_image.save('inverted_custom_channels.jpg')
```

By applying various inversion techniques to images generated by the DALL-E 2 API, you can create unique and visually engaging images, adding variety and interest to your projects.

Coding Exercise

In this assignment, you will be performing a series of image processing tasks using DALL-E and the Python Imaging Library (PIL). You will generate a unique image, apply a rotation transformation, and perform a color inversion.

The steps are as follows:

Use DALL-E to generate a visually appealing image using the provided prompt. Save the image as `cool.png`.
Rotate the generated image by 45 degrees, demonstrating your understanding of basic transformations in image processing. Save the rotated image as `rotated_45_cool.png`.
Finally, invert the colors of the rotated image. This step will familiarize you with color space transformations. Save the final image as `final_cool.png`.