Learning Objectives

Learners will be able to...

- **Understand the concept of image composition and learn how to combine multiple images to create more complex scenes**
- **Apply built-in image filters from the PIL library to enhance or alter the appearance of images generated by the DALL-E 2 API**
- **Create custom image transformations using the ImageOps module from the PIL library to achieve unique effects**
- **Combine advanced image manipulation techniques with the DALL-E 2 API to generate a wide variety of artistic images**
- **Experiment with various image manipulations and compositions to create custom images tailored to your specific needs or preferences**

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.

Image Composition

We learned how to generate images with DALL-E 2 API and manipulate basic properties like size, aspect ratio, brightness, contrast, saturation, and hue. In this lesson, we will explore more advanced image manipulation techniques to create even more variations and artistic effects.

Before that we are going to start with the same steps which are importing our libraries and having our function that generates our prompts.

```
import os
import openai
import secret
from PIL import Image
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']
```

Image setting up

One technique to create more complex images is by composing multiple images together. In this example, we will overlay a generated image of a moon on top of an image of a night sky.

First step we are going to get our function to generate the images and save them into new files.

```
# Generate moon and night sky images
moon_image_url = generate_base_image('moon')
night_sky_image_url = generate_base_image('night sky with
        stars')

img_data = requests.get(moon_image_url).content
with open('moon_image.jpg', 'wb') as handler:
    handler.write(img_data)

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

If you don't like the pictures generated, generate them again. If you like one and not the other comment out the code generation for the one you want. If you are happy with both, please comment out the generation code so we don't have to waste time generating images again and saving them over the image you want. At the end you should have something more or less like this :

```
"""
# Generate moon and night sky images
moon_image_url = generate_base_image('moon')
night_sky_image_url = generate_base_image('night sky with
        stars')

img_data = requests.get(moon_image_url).content
with open('moon_image.jpg', 'wb') as handler:
    handler.write(img_data)

img_data = requests.get(night_sky_image_url).content
with open('night_sky_image.jpg', 'wb') as handler:
    handler.write(img_data)
"""
```

Now we are ready to combine our images.

Image Overlay

Image composition

The first step is to open our saved images as variables.

```
moon_image=Image.open('moon_image.jpg')
night_sky_image=Image.open('night_sky_image.jpg')
```

The next step is to resize our images.

```
# Resize the moon image to fit the composition
moon_image = moon_image.resize((200, 200), Image.ANTIALIAS)
```

Resize the moon image: The `resize()` function is used to change the size of the moon image. The target size is specified as a tuple of (width, height), which in this case is `(200, 200)`. The `Image.ANTIALIAS` parameter is used to apply a high-quality downsampling filter that smooths out the image when resizing.

```
# Overlay the moon image on top of the night sky image
night_sky_image.paste(moon_image, (150, 100), moon_image)
```

Overlay the moon image on the night sky image: The `paste()` function is used to overlay the moon image on top of the night sky image. The function takes three arguments: the image to paste `moon_image`, the position where the pasted image should be placed (specified as an `(x, y)` tuple), and an optional `mask` (`moon_image` in this case). The mask is used to preserve transparency in the pasted image, which is important when overlaying images with transparent backgrounds. If an error was generated please read the note section, otherwise you can continue to the save image section.

Please note: if you get a bad transparency mask error, it is because the image you are trying to paste does not have an alpha channel. An **alpha channel** is a layer of transparency information that is stored in an image file. It allows parts of the image to be transparent, so that other images can be seen behind them.

In your code, you are trying to paste the moon image onto the night sky image, but the moon image does not have an alpha channel. This means that the Python Imaging Library (PIL) does not know how to blend the two images together, and it throws an error.

To fix this error, you need to convert the moon image to a format that supports alpha channels. You can do this using a graphics editing program like Photoshop or GIMP. We will do it using the `convert("RGBA")` function. Once you have converted the moon image, you can paste it onto the night sky image without any errors.

```
# Convert the moon image to RGBA format
moon_image = moon_image.convert("RGBA")

# Paste the moon image onto the night sky image
night_sky_image.paste(moon_image, (150, 100), moon_image)
```

Now we need to save our new image

```
# Save the composed image to a file
night_sky_image.save('night_sky_with_moon.jpg')
```

Save the composed image to a file: The `save()` function is used to save the composed image to a file. The file name is specified as a string (`night_sky_with_moon.jpg`), and the image will be saved in the same folder as your script.

Here is another approach to check the mode of the `moon_image` variable. Firstly, we check if it is in the right format. If it is so, we paste the image. Else we convert it to `"RGBA"` and then run the `paste` function.

```
# Overlay the moon image on top of the night sky image
if moon_image.mode in ('RGBA', 'LA') or (moon_image.mode == 'P'
        and 'transparency' in moon_image.info):
    night_sky_image.paste(moon_image, (150, 100))
else:
    moon_image = moon_image.convert("RGBA")
    night_sky_image.paste(moon_image, (150, 100))
```

If for example you wanted the image at the top right corner it would be `(0,0)` , you can play around with the coordinates so it fits your image better.
Depending on your image for now, you will basically see an overlay of your moon into your night sky. In most cases, using this method transparency will not work that well.

Filters

Now we will explore how to apply various image filters to images generated by the DALL-E 2 API using the Python Imaging Library (PIL). Image filters can be used to enhance, stylize, or transform your images, creating unique and visually striking results.

For now we are going to go back to our apple example.

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


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

Run it once to generate your “apple” then comment out the code so we have the base apple that we want to interact with. Your comment should more or less look like this

```
"""
base_image_url = generate_base_image('red apple')


img_data = requests.get(base_image_url).content
with open('base_apple.jpg', 'wb') as handler:
    handler.write(img_data)
"""
```

The last step is saving our image as a variable so we can more easily interact with it.

```
apple_image=Image.open('base_apple.jpg')
```

PIL

The PIL library provides several built-in filters that you can apply to your images. These filters include blurring, sharpening, contouring, edge detection, and many others. We will explore some of these filters and learn how to apply them to images generated by the DALL-E 2 API.

Gaussian Blur

A Gaussian blur filter softens the image by reducing noise and details. This filter can be used to create a dreamy or ethereal effect, or to reduce noise in an image.

```
from PIL import ImageFilter

# Apply a Gaussian blur filter to the base image
blurred_image =
        apple_image.filter(ImageFilter.GaussianBlur(radius=5))

# Save the blurred image to a file
blurred_image.save('blurred_red_apple.jpg')
```

Contour

The contour filter traces the edges of objects in the image, creating a high-contrast image. It is a filter that enhances the edges and boundaries within an image, making them more pronounced. It detects areas with rapid changes in pixel intensity and applies a highlighting effect to the edges. The resulting image will have a stylized appearance, emphasizing the contours of objects in the image. It is usually a black-and-white image that highlights the shapes in the scene.

```
from PIL import ImageFilter

# Apply a contour filter to the base image
contour_image = apple_image.filter(ImageFilter.CONTOUR)

# Save the contour image to a file
contour_image.save('contour_red_apple.jpg')
```

Edges

In the context of image processing, "**edges**" refer to boundaries or transitions between different regions in an image. These boundaries usually occur at points where there is a sharp change in color or intensity values. Identifying edges is a crucial step in many computer vision and image processing tasks, as it helps in understanding the structure and objects present in an image. In the next example, we are using filters that focus on identifying and enhancing the edges in the image.

Edge

The edge enhance filter highlights the edges of objects in the image, making them more distinct and pronounced. This filter can be used to create a more stylized or artistic effect.

```
from PIL import ImageFilter

# Apply an edge enhance filter to the base image
edge_enhanced_image =
        apple_image.filter(ImageFilter.EDGE_ENHANCE)

# Save the edge enhanced image to a file
edge_enhanced_image.save('edge_enhanced_red_apple.jpg')
```

Find Edges

The find edges filter is another type of image filter that enhances the edges in an image, similar to the contour filter. However, the find edges filter typically produces a more pronounced and higher contrast effect, making the edges stand out even more.

```
from PIL import ImageFilter

# Apply a find edges filter to the base image
edge_image = apple_image.filter(ImageFilter.FIND_EDGES)

# Save the edge image to a file
edge_image.save('edge_red_apple.jpg')
```

In this lesson, we explored various image filters provided by the PIL library and learned how to apply them to images generated by the DALL-E 2 API. By combining these filters with the DALL-E 2 API, you can create a wide

array of unique and visually striking images.

Feel free to experiment with different filters and combinations to create your own custom images using the DALL-E 2 API and the PIL library.

Coding Exercise

Assessment Question

1. Using OpenAI's DALL-E 2, generate an image of a plane in mid-flight. Save the generated image as `plane.png`.
2. Utilize the Python Imaging Library (PIL) to create an edge-enhanced version of the image you generated in the previous step. Save the enhanced image as `enhanced_plane.png`.

In this task, you will use OpenAI's DALL-E 2 to generate an image based on a textual description and then apply an image processing operation using PIL. The task involves two major steps:

1. **Image Generation with DALL-E 2**: Using OpenAI's DALL-E 2, write a function `generate_base_image(prompt)` that takes a text prompt as input and returns a URL for an image generated by DALL-E 2. The image should be 512x512 pixels in size. Download the image and save it as `plane.png`.
2. **Image Processing with PIL**: Next, load the `plane.png` image and apply an edge enhance filter using PIL's `ImageFilter.EDGE_ENHANCE`. Save the resulting edge-enhanced image as `enhanced_plane.png`.