Learning Objectives

Learners will be able to...

- **Generate a response using `openai.Completion.create`**
- **Get familiar with PIL imaging library**
- **Generate an image and manipulate it using PIL**
- **Learn about the different parameters of the DALL-E API call**

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.

Text to Image

PIL

PIL (Python Imaging Library) is a popular third-party library used for working with images in Python. It provides a wide range of functions to handle different image processing tasks, such as opening and saving images in various file formats, cropping and resizing images, applying filters and transformations, and much more. In this module, we will use the PIL library to manipulate the image we generated.

DALL-E

In the previous lessons, we discussed how to generate images using DALL-E 2. Now, we will dive into the exciting world of image manipulation, enabling you to create a wide variety of variations from the original generated image.

> **DALL-E** 2 is a powerful AI model developed by OpenAI that can generate high-quality images from textual descriptions. However, the API is not limited to just generating images - it also allows you to manipulate the generated images by adjusting certain parameters. We will explore how to change image size, aspect ratio, brightness, contrast, saturation, hue, and other properties using the DALL-E 2 API.

Generating Image

Let's get started with a simple code example that demonstrates how to adjust some of these parameters. In this example, we will generate an image of a "red apple" and then manipulate its properties to create different variations.
In order to generate an image using the DALL·E we can use the `openai.Image.create` call. This time we are going to create a function that will take our prompt as an argument and manipulate it from there. We are going to import our libraries and API key then we are going to create our 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']

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

In order to not have to keep a copy and past our result in, we are going to save the image inside the URL as a file in our Codio box.

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

After generating our image we are going to comment out the function for the code generation. Since our image is saved in a file we can just interact with that now. We can use multi-line comments using the following syntanx `"""  """`.

Image Manipulation

Image Manipulation Function

We first generated a base image of a “red apple” using the DALL-E 2 API. We saved our image as `image_name.jpg`. I know really creative. Now we are going to access that image and start manipulating it. The first step we are going to do is create a variable that points to our first image and calls that `base_image`.

```
"""
base_image_url = generate_base_image('red apple')
"""
base_image = Image.open('image_name.jpg')
```

Now we are ready to create a function that will interact with the `PIL` library. The function takes a couple of variables as arguments. Lets start by defining what those are.

```
def manipulate_image(image, size, aspect_ratio, brightness,
        contrast, saturation, hue)
```

1. **image**: A PIL image object representing the base image that you want to manipulate.
2. **size**: A float value between 0 and 1, representing the scale factor by which you want to resize the image. A value of 1 will keep the original size, while a value of 0.5 will reduce the size by half, and a value of 0.75 will resize the image to 75% of its original size.
3. **aspect_ratio**: A float value representing the aspect ratio for resizing the image. A value of 1 will maintain the original aspect ratio, while values greater than 1 will stretch the image vertically and values less than 1 will stretch the image horizontally.
4. **brightness**: A float value representing the factor by which to adjust the image's brightness. A value of 1 means no change, while values greater than 1 will increase brightness, and values less than 1 will decrease brightness.
5. **contrast**: A float value representing the factor by which to adjust the image's contrast. A value of 1 means no change, while values greater than 1 will increase contrast, and values less than 1 will decrease contrast.
6. **saturation**: A float value representing the factor by which to adjust the image's color saturation. A value of 1 means no change, while values greater than 1 will increase saturation, and values less than 1 will decrease saturation.
7. **hue**: A float value representing the amount by which to adjust the image's hue. A value of 0 means no change, while positive and negative values will rotate the hue in different directions.

```
# Manipulate the image properties
def manipulate_image(image, size, aspect_ratio, brightness,
        contrast, saturation, hue):
    # Resize the image
    new_size = (int(image.width * size), int(image.height *
        aspect_ratio * size))
    resized_image = image.resize(new_size, Image.ANTIALIAS)


    # Adjust the image properties
    from PIL import ImageEnhance, ImageOps
    enhanced_image =
        ImageEnhance.Brightness(resized_image).enhance(brightness)


    enhanced_image =
        ImageEnhance.Contrast(enhanced_image).enhance(contrast)
    enhanced_image =
        ImageEnhance.Color(enhanced_image).enhance(saturation)


    # Adjust hue using ImageOps module
    enhanced_image =
        ImageOps.colorize(enhanced_image.convert('L'), 'black',
        'white', midpoint=128 - int(128 * hue))


    return enhanced_image
```

Run the following to make sure the code run without error.

We define a function `manipulate_image()` that takes in the base image and various parameters for image manipulation. We use the Python Imaging Library (PIL) to resize the image, and adjust its brightness, contrast, saturation, and hue. We use `ImageOps.colorize()` to adjust the hue of the image.

```
 # Example of image manipulation
manipulated_image = manipulate_image(
  base_image, size=0.5, aspect_ratio=1, brightness=1.2,
  contrast=1.5, saturation=0.8, hue=0.1
)
```

Finally, we demonstrate how to use this function to create a manipulated version of the original image by adjusting the parameters.

Now we are going to save our manipulated image as `manipulated_red_apple.jpg`.

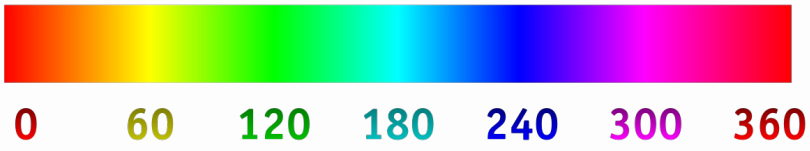
```
# Save the manipulated image to a file
manipulated_image.save('manipulated_red_apple.jpg')
```

Hue and Saturation

The **hue** parameter in the `manipulate_image()` function is used to shift the hue values of the image, resulting in different color schemes. The hue value is a float, typically ranging from -1 to 1, where 0 means no change, positive values shift the hue clockwise in the color wheel, and negative values shift the hue counterclockwise.

Hue values in color space are typically represented in a circular form, often called a “color wheel.” The color wheel is divided into 360 degrees, with different colors associated with different angle values:

Red: 0 degrees (or 360 degrees)
Yellow: 60 degrees
Green: 120 degrees
Cyan: 180 degrees
Blue: 240 degrees
Magenta: 300 degrees



Link

`n` the `manipulate_image()` function, we scale the hue parameter from -1 to 1 to cover the entire range of hue values. When the hue parameter is set to a specific value, the hue values in the image are shifted by that fraction of the full hue range (360 degrees).

For example:

- A hue value of 0.5 would shift the hue values by 180 degrees clockwise in the color wheel, effectively changing red to cyan, green to magenta, and blue to yellow.
- A hue value of -0.5 would shift the hue values by 180 degrees counterclockwise in the color wheel, effectively changing red to cyan, green to magenta, and blue to yellow (same as the previous example, since the shift is by 180 degrees).
- A hue value of 0.33 would shift the hue values by 120 degrees clockwise in the color wheel, effectively changing red to green, green to blue, and blue to red.
- A hue value of -0.25 would shift the hue values by 90 degrees counterclockwise in the color wheel, effectively changing red to yellow,

green to cyan, and blue to magenta.

You can experiment with different hue values to achieve various color schemes for your image. Keep in mind that the color shifts might be more complex in real images, as they contain a mix of colors and may not always result in the exact colors mentioned in the examples.

Saturation

Saturation is a color property that represents the intensity or purity of a color. It describes how vivid or dull a color appears. In the context of image processing, adjusting the saturation of an image can make the colors in the image appear more vibrant or more muted.

In the HSV (Hue, Saturation, and Value) color model, saturation is represented as a percentage (0% to 100%) or a value ranging from 0 to 1:

0% saturation (or a value of 0) means no color, resulting in a grayscale image.
100% saturation (or a value of 1) means the colors are fully saturated, appearing in their most intense form.

In our function, the saturation parameter is a float value that determines the factor by which the saturation of the image is adjusted:

- A saturation value of 1 means no change to the saturation of the image.
- Saturation values greater than 1 will increase the saturation of the image, making the colors appear more vivid and intense.
- Saturation values less than 1 and greater than 0 will decrease the saturation of the image, making the colors appear more muted or washed out.
- A saturation value of 2 would double the saturation of the colors in the image, making them appear more vibrant.
- A saturation value of 0.5 would reduce the saturation of the colors by half, making them appear more muted.
- A saturation value of 0 would remove all color from the image, resulting in a grayscale image.

When manipulating images, adjusting the saturation can have a significant impact on the overall appearance and mood of the image. Experimenting with different saturation values can help you achieve the desired look for your image.

Trying Out Different Color Variations

Keep color

Let's try creating an extra variation. This time we are going to keep the color the same. Try out variations.

To maintain the original color when the hue is 0. Add the code into the `manipulate_image()` function:

```
# Adjust hue only if hue is not 0
if hue != 0:
    # Convert image to HSV
    hsv_image = enhanced_image.convert('HSV')
    # Shift hue value
    hsv_image = hsv_image.point(lambda p: (p + int(256 * hue)) %
        256 if p < 256 else p)
    # Convert back to RGB
    enhanced_image = hsv_image.convert('RGB')
```

The `ImageOps.colorize()` function used in the code is converting the image to grayscale. We first check if the hue parameter is not 0 before adjusting the hue of the image. If the hue is 0, the color of the image remains unchanged.

In this example, we've changed the size parameter to 0.8 (80% of the original size) and kept the aspect ratio at 1 to maintain the original aspect ratio. The brightness, contrast, saturation, and hue parameters are set to their default values to keep the color properties unchanged. Let's try a another example,

```
# Example of image manipulation with only size change
size_changed_image = manipulate_image(
    base_image, size=0.8, aspect_ratio=1, brightness=1,
    contrast=1, saturation=1, hue=0
)

# Save the size-changed image to a file
size_changed_image.save('size_changed_red_apple.jpg')
```

The resulting image will be saved as `size_changed_red_apple.jpg` in the same folder as your script.

Purple

Now let's try making it more purple.

```
# Example of image manipulation with a more purple hue
purple_image = manipulate_image(
    base_image, size=1, aspect_ratio=1, brightness=1,
    contrast=1, saturation=1, hue=0.8
)

# Save the purple image to a file
purple_image.save('purple_red_apple.jpg')
```

In this example, we set the hue parameter to 0.8, which shifts the hue towards the purple range. The size, aspect ratio, brightness, contrast, and saturation parameters are set to their default values to maintain the original properties of the image.

The resulting image will be saved as 'purple_red_apple.jpg' in the same folder as your script. Please note that the hue value of 0.8 is just an example, and you may need to adjust the value to achieve the desired shade of purple.

Try the purple image with different combination of saturation. For example, try it with the following : `0`, `0.5` , `2` .

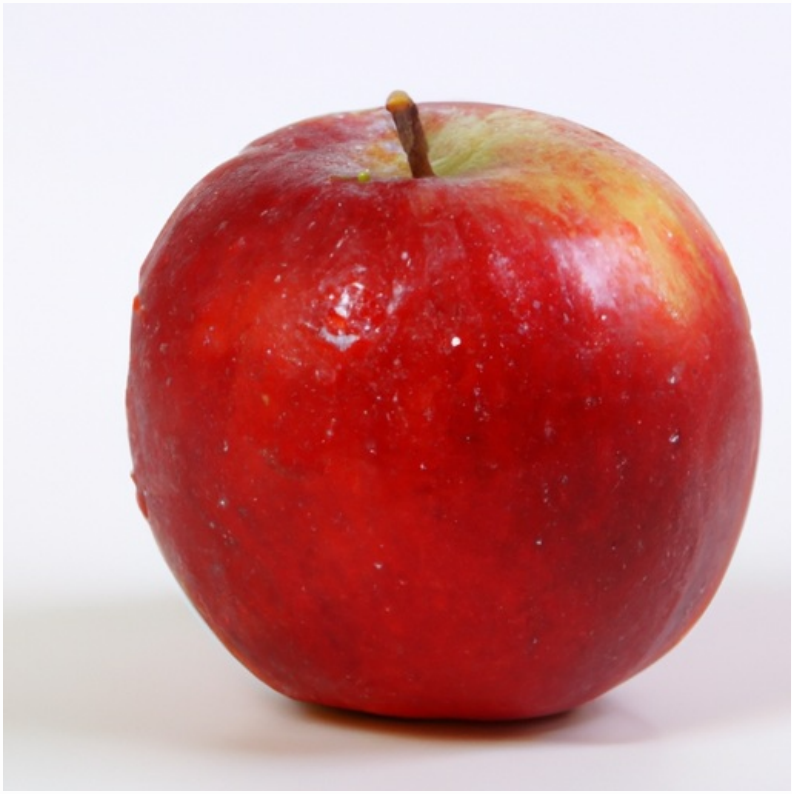
We have demonstrated how to create variations of an image using the `manipulate_image()` function. You can use this function to create different versions of an image with varying size, aspect ratio, brightness, contrast, saturation, and hue.

To create a version of the image with the original color, set the hue parameter to 0. In the provided code, we first check if the hue parameter is not 0 before adjusting the hue of the image. If the hue is 0, the color of the image remains unchanged.

We have also provided examples of how to create a resized version of the image, as well as a version with a more purple hue. You can experiment with different combinations of parameters to achieve the desired look for your image.

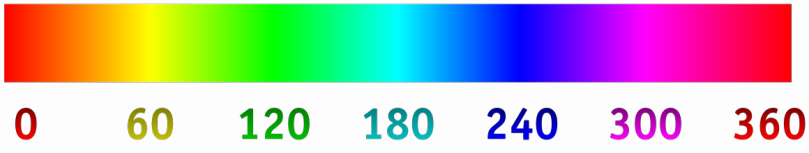
Coding Exercise

Given the following base image and code:



To help remember that:

Red: 0 degrees (or 360 degrees)
Yellow: 60 degrees
Green: 120 degrees
Cyan: 180 degrees
Blue: 240 degrees
Magenta: 300 degrees



Link