Learning Objectives

Learners will be able to...

- **Develop a program that interacts with DALL-E**
- **Create images using Dall-E**
- **Apply filters to images and resize images using Dall-E**

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.

GitHub

Connecting to GitHub

In this lab, you are going to build a simple program and integrate Dall-E to it. In order to showcase this project for your portfolio, you are going to use GitHub. If you do not yet have an account, please create one now. We are going to clone a repo that will contain the code for your Chatbot.

Connecting GitHub and Codio

You need to connect GitHub to your Codio account. This only needs to be done one time.

- In your Codio account, click on your username
- Click on **Applications**

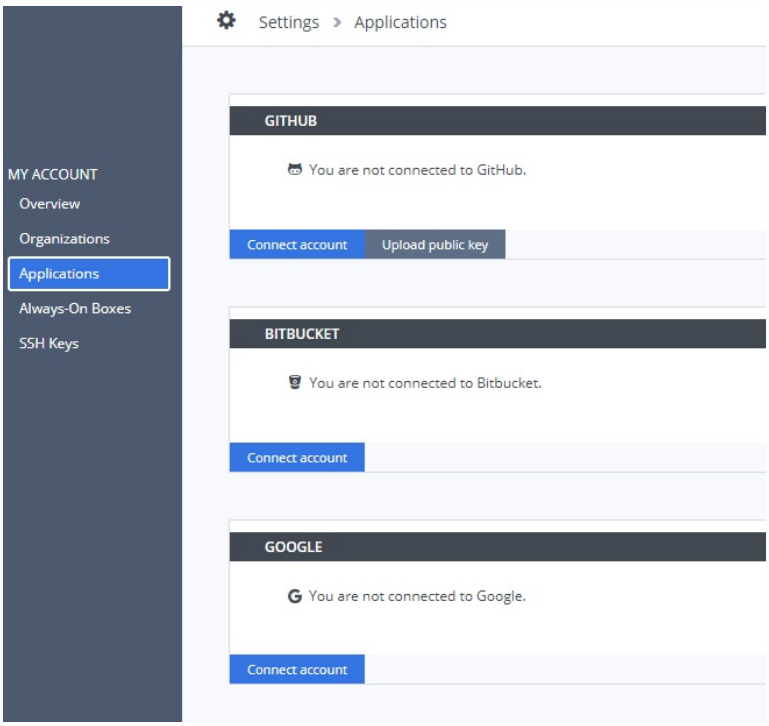
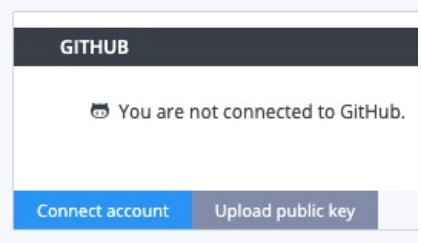


Image showing Codio account bar. The word Application is highlighted

- Under GitHub, click on **Connect account**



an image displaying the words connect account in blue and upload public key in gray

- You will be using an SSH connection, so you need to click on **Upload public key**

Fork the Repo

- Go to the Dall-E lab repo. This repo is the starting point for your project.
- Click on the `Fork` button in the top-right corner.
- Click `Create fork` green button.
- Click the `Code` button.
- Copy the **SSH** information. It should look something like this:

```
git@github.com:<your_username>/Dall-E_lab.git
```

info

Important

If you do not use the SSH information, you will have to provide your username and Personal Access Token (PAT) to GitHub each time you push or pull from the repository. See this documentation for setting up a PAT for your GitHub account.

In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:<your_username>/Dall-E_lab.git
```

- You should see a `Dall-E_Lab` directory appear in the file tree.

On the last page, we'll show you how to push to GitHub.

You are now ready for the lab!

Creating Main Function

Goals:

This lab will have learners generate images using prompts given to DALL-E, and then manipulate those images with PIL. They will need to understand how to interact with DALL-E and the PIL library. We will write code to generate images, and then write code to modify those images.

There will be **Try It!** buttons that will call our function in our terminal (bottom-left panel).
The first step is to have all the libraries and API keys:

```
#import our libraries
import os
import openai
from PIL import Image,ImageOps,ImageFilter
from io import BytesIO
import requests


# Set API key and prompt
openai.api_key = os.getenv('OPENAI_KEY')
```

Let's create a main function that acts as a control center for the program. Here's a simple example to give you an idea.

```
def main():
    while True:
        print("\n1. Generate an image with DALL-E")
        print("2. Create variations of an image")
        print("0. Exit")

        option = input("Choose an option: ")

        if option == "1":
            prompt = input("Enter the prompt for DALL-E: ")
            # Generate the image with DALL-E
            generate_image(prompt)
        elif option == "2":
            image_path = input("Enter the path of the image: ")
            # Open the image file
            create_variations(image_path)
        elif option == "0":
            print("Exiting the program...")
            break
        else:
            print("Invalid option. Please enter a valid
        number.")
```

In this program, the main function is running a continuous loop that prompts the user to choose an option. If the user chooses option “1”, they’re asked for a prompt which is then passed to the `generate_image` function (where DALL-E would be used to generate an image). If the user chooses option “2”, they’re asked for an image path, and the image at that path is opened and passed to the `create_variations` function. Lastly, an option “0” is given to exit our program.

Next we are going to put placeholder code for the function code

```
def generate_image(prompt1):
    # Generate the image with DALL-E
    print(f"Generating an image for the prompt: {prompt1}")
    # Call DALL-E API here

def create_variations(img):
    # Create variations of an image using PIL
    print("Creating variations of the image...")
     # Call DALL-E API here

if __name__ == "__main__":
    main()
```

The `generate_image` and `create_variations` functions are placeholders for you to implement the actual logic of interacting with DALL-E.

generate_image

Now we are going to write the first piece of our program which is the program to generate our image. All the code on this page should go under the comment `# call DALL-E API here`

```
def generate_image(prompt1):
    # Generate the image with DALL-E
    print(f"Generating an image for the prompt: {prompt1}")
     # Call DALL-E API here
```

We are going to fill our function. You can create an image using DALL-E by providing a textual prompt, the number of images you want, and the desired size of the image. We are going to make our generic function create 1 image, with the size 256x256.

```
response = openai.Image.create(
  prompt=prompt1,
  n=1,
  size="256x256"
)
```

Test out that everything runs without error before moving to the next steps.

DALL-E generates the image and returns a response that includes a URL where the image can be accessed. You can extract this URL from the response. We already know what the URL looks like, let's extract it to a variable we can use.

```
image_url = response['data'][0]['url']
```

Now that you have the URL of the image, you can download it using the requests library. After sending a `GET` request to the image URL and receiving the image data, you can write this data to a file to save the image locally. For our program we will save the file generated as `image_created.png`.

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

As an indication that our function got done running we will add a print message at the end of our code.

```
print("Your image was generated under the `image_created.png`
file name.")
```

The following link will open the `image_created.png` on the bottom left panel.

By following these steps, you can generate a variety of images from textual prompts using DALL-E, illustrating the power of AI in creative tasks like image generation. Just like that we have our first option ready.

Variation

The transformative capability of OpenAI's DALL-E extends beyond generating images from scratch; it can also generate variations of existing images. This capacity enriches the creative process, providing novel perspectives and interpretations of a given subject, or assisting in creating different versions of the same concept. For our second option we are going to interact with the following function:

```
def create_variations(img):
    # Create variations of an image using PIL
    print("Creating variations of the image...")
    # Call DALL-E API here
```

The Images API offers three methods of interaction with images, one of which is creating variations of an existing image. The API's image variation endpoint enables the generation of images similar to the input image. This feature is particularly useful when you have a base image and want to create variations without manually editing the image.

The code to generate a variation of an image is similar to that used for generating an image from scratch, with a few minor differences:

Instead of providing a textual prompt as we did while creating an image from scratch, we give DALL-E an existing image as a prompt. The image parameter in `openai.Image.create_variation` expects an open image file in read-binary mode ("rb"). Make sure the image is located locally, or will have to provide the whole path.

```
response = openai.Image.create_variation(
  image=open(img, "rb"),
  n=1,
  size="256x256"
)
```

Just like in the image creation process, DALL-E returns a URL for the created variation image, which you can extract from the response.

```
image_url = response['data'][0]['url']
```

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

Just to make everything more clear you can try adding a print statement at the end to indicate your function finished running.

```
print("Your image was generated under the `image_var.png` file
        name.")
```

By employing DALL-E's image variation capabilities, you can diversify your image portfolio, generate fresh ideas, or improve the original design.

Changing Size

Now let's try adding additional options to our program. We are going to add a function that can resize our images for us. You can resize images using the Python Imaging Library (PIL). First we are going to add a print statement after option 2.

```
def main():
    while True:
        print("\n1. Generate an image with DALL-E")
        print("2. Create variations of an image")
        print("3. Resize an image")
        print("0. Exit")
```

We also need to add a resize option call. Place the following code after option 2.

```
elif option=="3":
          image_path = input("Enter the path of the image you
        want resized: ")
          # run resize function
          re_size(image_path)
```

A Try It! button is not provided here since we have not written our resize function yet and because of that you would just get an error.

Let's write our `re_size` function. We know that for our Dall-E we need our images to be in the following 3 dimensions: 256x256, 512x512, or 1024x1024. We will ask for additional user input as to pick which one

```
def re_size(image_path):
## Open an image file
    with Image.open(image_path) as img:
        # Ask the user to choose a dimension
        print("Choose a dimension for resizing the image:")
        print("1. 256x256")
        print("2. 512x512")
        print("3. 1024x1024")
        option = input("Enter your option: ")
        if option == "1":
            target_size = (256, 256)
        elif option == "2":
            target_size = (512, 512)
        elif option == "3":
            target_size = (1024, 1024)
        else:
            print("Invalid option. Using default size 256x256.")
            target_size = (256, 256)

        # Resize the image
        resized_img = img.resize(target_size)

        # Save the resized image
        resized_img.save('resized_image.png')
```

We're asking the user to choose a dimension for resizing the image. If the user's choice is not among the three options (1, 2, or 3), we default to the size 256x256. After the user makes their choice, we resize the image to the selected dimension and save it as `'resized_image.png'`.

Let's add a little message saying the function ended.

```
print("Your given image was resized and saved under
        `resized_image.png` ")
```

Filters

The last option we are going to cover in our lab is to create a function that creates a filter of a given image. Let's start by adding our print statement at the top of our `main` function.

```
print("4. Add a filter to your image")
```

Then we are going to create an extra option after option 3. Let's call it option 4.

```
elif option=="4":
          image_path = input("Enter the path of the image you
        want to add a filter to: ")
          # run filter function
          filters(image_path)
```

Now we are ready to add our extra function that adds filters using PIL. Our function should ask the user to pick which filter they want to apply. The options are :
1. `contour`: The contour filter traces the edges of objects in the image, creating a high-contrast, black-and-white image that highlights the shapes in the scene.
1. `EDGE_ENHANCE`: 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.
1. `FIND_EDGES`: The find edges filter detects the edges of objects in the image and creates a high-contrast, black-and-white image that highlights these edges.

First, our function should open an image file. Then we prompt the user to choose one of the three provided filters to apply to the image. If the user's input is not among the three options (1, 2, or 3), we default to the Contour filter. After applying the chosen filter, we save the result to `filtered_image.jpg` in the current directory.

```
def filters(image_path):
    # Open an image file
    with Image.open(image_path) as img:
        # Ask the user to choose a filter
        print("Choose a filter to apply on the image:")
        print("1. Contour")
        print("2. Edge Enhance")
        print("3. Find Edges")
        option = input("Enter your option: ")

        if option == "1":
            filtered_img = img.filter(ImageFilter.CONTOUR)
        elif option == "2":
            filtered_img = img.filter(ImageFilter.EDGE_ENHANCE)
        elif option == "3":
            filtered_img = img.filter(ImageFilter.FIND_EDGES)
        else:
            print("Invalid option. Applying Contour filter by
        default.")
            filtered_img = img.filter(ImageFilter.CONTOUR)

        # Save the filtered image
        filtered_img.save('filtered_image.png')
        print("Filter applied successfully, 'filtered_image.png'
        created.")
```

Coding Exercise

The **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. We have interacted with it in our previous assignment.

To implement the Gaussian blur in PIL, we use the filter function with `ImageFilter.GaussianBlur`. As an example, here's a code snippet that applies a Gaussian blur filter to an image with a radius of 5:

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

For our coding exercise, you are going to add an extra option to your `filter` function. That function should apply Gaussian Blur to a given image, if the user selects option 4. As a bonus feel free to add a 5th option that shows your current directory's files.

GitHub Push

Pushing to GitHub

For the final step, you must push your work to GitHub. For that we need to move our main file into our Dall-E_lab folder's main file (cloned from GitHub).

In the terminal, after you are done copying the main file:
* Move into the Dall-E lab folder

```
cd Dall-E_lab
```

- Commit your changes:

```
git add .
git commit -m "Finished The Dall-E program"
```

- Push to GitHub:

```
git push
```