❖ **Prerequisites:**

- Python
- Visual Studio Code
- Installation of:all libraries needed, Jupyter extension

Solutions for Exercises:

"The modified code errors are also available in the text files in the link"

Exercise 1:

1. **It does not print the fruit at the correct index, why is the returned result wrong?**

The returned result is wrong because sets in Python do not maintain the order of elements. When iterating over a set, the order of elements is arbitrary and does not correspond to the original order of insertion. Thus, indexing into a set based on a specific integer value does not guarantee that you will get the expected element at that index.

2. **How could this be fixed?**

To fix this, we should use a data structure that maintains the order of elements, such as a list. By passing a list of fruits instead of a set, we can ensure that the elements are accessed in the correct order based on their indices.

Exercise 2:

1. **Can you spot the obvious error?**

The obvious error in the original implementation is that the assignment within the **swap** function does not swap the coordinates properly. Specifically, the line:

coords[:, 0], coords[:, 1], coords[:, 2], coords[:, 3], = coords[:, 1], coords[:, 1], coords[:, 3], coords[:, 2]

does not correctly swap **x** and **y** coordinates. It incorrectly assigns values, causing duplicate and misplaced coordinates.

2. **After fixing the obvious error it is still wrong, how can this be fixed?**

Even after fixing the obvious error, we must correctly swap the **x** and **y** coordinates for both pairs **(x1, y1)** and **(x2, y2)**.

We need to create a temporary array to hold the original values during the swap.

Exercise 3:

1. **For some reason the plot is not showing correctly, can you find out what is going wr ong?**

in line f = open("data_file.csv", "w") in the generated file creates a blank line between each line of data. When these lines are read from the file and arranged in the list, it produces a series of empty sublists that the program gives an error.

2. **How could this be fixed?**

Solution: f = open("data_file.csv", "w+", newline='').

For Exercise 4:

1. **Changing the batch_size from 32 to 64 triggers the structural bug.**

The structural bug arises from a mismatch in tensor sizes when concatenating real and generated samples. The bug manifests because the number of real samples may not always be a multiple of the batch size. Here's a detailed explanation of the issue and the fix:

When the batch_size is 64, the number of real samples in the last batch of an epoch may be less than 64 if the dataset size is not perfectly divisible by 64.

When concatenating real and generated samples, this leads to a mismatch in tensor sizes.

Code Causing the Bug:

```
# Data for training the discriminator

real_samples_labels = torch.ones((batch_size, 1)).to(device=device)

latent_space_samples = torch.randn((batch_size, 100)).to(device=device)

generated_samples = generator(latent_space_samples)

generated_samples_labels = torch.zeros((batch_size, 1)).to(device=device)

all_samples = torch.cat((real_samples, generated_samples))

all_samples_labels = torch.cat((real_samples_labels, generated_samples_labels))
```

Fixed Code:

Adjust the labels to match the actual size of real_samples instead of assuming it to be equal to batch_size.

```
# Data for training the discriminator

real_samples = real_samples.to(device=device)

real_samples_labels = torch.ones((real_samples.size(0), 1)).to(device=device)  # Adjust label size

latent_space_samples = torch.randn((real_samples.size(0), 100)).to(device=device)

generated_samples = generator(latent_space_samples)
```

generated_samples_labels = torch.zeros((real_samples.size(0), 1)).to(device=device) # Adjust label size

all_samples = torch.cat((real_samples, generated_samples))

all_samples_labels = torch.cat((real_samples_labels, generated_samples_labels))

2. **Can you also spot the cosmetic bug?**

The cosmetic bug is related to the incorrect indexing in the condition for displaying the generated images and losses. Specifically, the condition if n == batch_size - 1: is incorrect because it should be checking against the iteration number, not the batch size.

The condition to display images and losses is based on n == batch_size - 1, which does not properly trigger at the end of each batch.

Fixed Code:

Change the condition to if n == len(train_loader) - 1: to ensure it triggers at the end of each epoch.

Show loss and samples generated

if n == len(train_loader) - 1: # Corrected condition

    name = f"Generate images\n Epoch: {epoch} Loss D.: {loss_discriminator:.2f} Loss G.: {loss_generator:.2f}"

    generated_samples = generated_samples.detach().cpu().numpy()

    fig = plt.figure()

    for i in range(16):

        sub = fig.add_subplot(4, 4, 1 + i)

        sub.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")

        sub.axis('off')

    fig.suptitle(name)

    fig.tight_layout()

    clear_output(wait=False)

    display(fig)

1. **Structural Bug**: Adjust the size of the labels for real and generated samples to match the actual size of real_samples instead of assuming it to be equal to batch_size.

2. **Cosmetic Bug**: Correct the condition for displaying generated images and losses to trigger at the end of each epoch by using if n == len(train_loader) - 1:.