Generate print hello world using rot13 Close

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
```

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 64*64*3),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        return img.view(-1, 3, 64, 64)
```

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(64*64*3, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, img):
        img_flat = img.view(img.size(0), -1)
        return self.model(img_flat)
```

```
import torch.nn as nn

class Discriminator(nn.Module):
    def __init__(self, num_classes=2):
        super(Discriminator, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 4, 2, 1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Flatten()
        )
```

```
            self.classifier = nn.Sequential(
                nn.Linear(128 * 16 * 16, 1024),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Linear(1024, num_classes)
            )

        def forward(self, x):
            x = self.features(x)
            x = self.classifier(x)
            return x



    classifier = Discriminator(num_classes=2).to(device)
    classifier.load_state_dict(torch.load("/content/drive/MyDrive/classifier_model.pth", map_location=device))
    classifier.eval()
```

```
    Discriminator(
      (features): Sequential(
        (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
        (1): LeakyReLU(negative_slope=0.2, inplace=True)
        (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
        (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (4): LeakyReLU(negative_slope=0.2, inplace=True)
        (5): Flatten(start_dim=1, end_dim=-1)
      )
      (classifier): Sequential(
        (0): Linear(in_features=32768, out_features=1024, bias=True)
        (1): LeakyReLU(negative_slope=0.2, inplace=True)
        (2): Linear(in_features=1024, out_features=2, bias=True)
      )
    )
```

```
    from PIL import Image
    import torchvision.transforms as transforms
    import torch

    # Define transform (should match training transform)
    transform = transforms.Compose([
        transforms.Resize((64, 64)),  # match model input
        transforms.ToTensor(),
    ])

    # Load image
    img_path = "/content/drive/MyDrive/lumpy_skin_dataset/normal/normal_2090.png"  # 🔁 change to your test image
    image = Image.open(img_path).convert("RGB")
    image = transform(image).unsqueeze(0).to(device)  # Add batch dimension

    # Predict
    classifier.eval()
    with torch.no_grad():
        output = classifier(image)
        predicted = torch.argmax(output, dim=1).item()

    # Print result
    if predicted == 0:
        print("✅ Prediction: INFECTED")
        print("📝 Recommendation: Consult a veterinarian. Isolate the infected animal.")
    else:
        print("⚠️ Prediction: NORMAL")
        print("📝 Recommendation: The skin appears healthy. No treatment required.")
```

```
    ⚠️ Prediction: NORMAL
    📝 Recommendation: The skin appears healthy. No treatment required.
```

```
    from PIL import Image
    import torchvision.transforms as transforms
    import torch

    # Define transform (should match training transform)
    transform = transforms.Compose([
        transforms.Resize((64, 64)),  # match model input
        transforms.ToTensor(),
    ])
```

```
# Load image
img_path = "/content/drive/MyDrive/lumpy_skin_dataset/infected/infected_1667.png"  # 🔁 change to your test image
image = Image.open(img_path).convert("RGB")
image = transform(image).unsqueeze(0).to(device)  # Add batch dimension

# Predict
classifier.eval()
with torch.no_grad():
    output = classifier(image)
    predicted = torch.argmax(output, dim=1).item()

# Print result
if predicted == 0:
    print("✅ Prediction: INFECTED")
    print("📝 Recommendation: Consult a veterinarian. Isolate the infected animal.")
else:
    print("⚠️ Prediction: NORMAL")
    print("📝 Recommendation: The skin appears healthy. No treatment required.")
```

```
✅ Prediction: INFECTED
📝 Recommendation: Consult a veterinarian. Isolate the infected animal.
```

```
from PIL import Image
import torchvision.transforms as transforms
import torch

# Define transform (should match training transform)
transform = transforms.Compose([
    transforms.Resize((64, 64)),  # match model input
    transforms.ToTensor(),
])

# Load image
img_path = "/content/drive/MyDrive/lumpy_skin_dataset/normal/normal_1847.png"  # 🔁 change to your test image
image = Image.open(img_path).convert("RGB")
image = transform(image).unsqueeze(0).to(device)  # Add batch dimension

# Predict
classifier.eval()
with torch.no_grad():
    output = classifier(image)
    predicted = torch.argmax(output, dim=1).item()

# Print result
if predicted == 0:
    print("✅ Prediction: INFECTED")
    print("📝 Recommendation: Consult a veterinarian. Isolate the infected animal.")
else:
    print("⚠️ Prediction: NORMAL")
    print("📝 Recommendation: The skin appears healthy. No treatment required.")
```

```
⚠️ Prediction: NORMAL
📝 Recommendation: The skin appears healthy. No treatment required.
```

```
from PIL import Image
import torchvision.transforms as transforms
import torch

# Define transform (should match training transform)
transform = transforms.Compose([
    transforms.Resize((64, 64)),  # match model input
    transforms.ToTensor(),
])

# Load image
img_path = "/content/drive/MyDrive/lumpy_skin_dataset/infected/infected_1616.png"  # 🔁 change to your test image
image = Image.open(img_path).convert("RGB")
image = transform(image).unsqueeze(0).to(device)  # Add batch dimension

# Predict
classifier.eval()
with torch.no_grad():
    output = classifier(image)
    predicted = torch.argmax(output, dim=1).item()
```

```
# Print result
if predicted == 0:
    print("✅ Prediction: INFECTED")
    print("📝 Recommendation: Consult a veterinarian. Isolate the infected animal.")
else:
    print("⚠️ Prediction: NORMAL")
    print("📝 Recommendation: The skin appears healthy. No treatment required.")
```

```
✅ Prediction: INFECTED
📝 Recommendation: Consult a veterinarian. Isolate the infected animal.
```

```
from PIL import Image
import torchvision.transforms as transforms
import torch

# Define transform (should match training transform)
transform = transforms.Compose([
    transforms.Resize((64, 64)),  # match model input
    transforms.ToTensor(),
])

# Load image
img_path = "/content/drive/MyDrive/lumpy_skin_dataset/infected/infected_965.png"  # 🔁 change to your test image
image = Image.open(img_path).convert("RGB")
image = transform(image).unsqueeze(0).to(device)  # Add batch dimension

# Predict
classifier.eval()
with torch.no_grad():
    output = classifier(image)
    predicted = torch.argmax(output, dim=1).item()

# Print result
if predicted == 0:
    print("✅ Prediction: INFECTED")
    print("📝 Recommendation: Consult a veterinarian. Isolate the infected animal.")
else:
    print("⚠️ Prediction: NORMAL")
    print("📝 Recommendation: The skin appears healthy. No treatment required.")
```

```
✅ Prediction: INFECTED
📝 Recommendation: Consult a veterinarian. Isolate the infected animal.
```

```
from PIL import Image
import torchvision.transforms as transforms
import torch

# Define transform (should match training transform)
transform = transforms.Compose([
    transforms.Resize((64, 64)),  # match model input
    transforms.ToTensor(),
])

# Load image
img_path = "/content/drive/MyDrive/lumpy_skin_dataset/normal/normal_1203.png"  # 🔁 change to your test image
image = Image.open(img_path).convert("RGB")
image = transform(image).unsqueeze(0).to(device)  # Add batch dimension

# Predict
classifier.eval()
with torch.no_grad():
    output = classifier(image)
    predicted = torch.argmax(output, dim=1).item()

# Print result
if predicted == 0:
    print("✅ Prediction: INFECTED")
    print("📝 Recommendation: Consult a veterinarian. Isolate the infected animal.")
else:
    print("⚠️ Prediction: NORMAL")
    print("📝 Recommendation: The skin appears healthy. No treatment required.")
```

```
Prediction: NORMAL
Recommendation: The skin appears healthy. No treatment required.
```