In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import pathlib
import random
from PIL import Image
import cv2
from sklearn.model_selection import train_test_split
```

In [2]:
```
for dirname, _, filenames in os.walk(r"C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-01ab65973fd487a6cee4c5af1551c42b264eec5abab46bffd7c307ffef647e11.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-0797945218a97d6e5251b4758a2ba1b418cbd52ce4ef46a3239e4b939bd9807b.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-0c826b6f4648edf507e0cafdab53712bb6fd1f04dab453cee8db774a728dd640.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-0ff5ba898a0ec179a25ca217af45374fdd06d606bb85fc29294291facad1776a.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-1289c57b571e8e98e4feb3e18a890130adc145b971b7e208a6ce5bad945b4a5a.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-16f7515e1d6aa6d7dd3af4bca38c8065bfab9d426c5fd75b3c4bc51d737fb9d0.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-1e80d2d6f3e9cf96c1cb33f3e47f3e5a3f4a6eb26fa3ab479d462e1ac837ba66.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-1f941001f508ff1bd492457a90da64e52c461bfd64587a3cf7c6bf1bcb35adab.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
s-20f5f654ae5fbcc405b465ce257c187f81eb5fc070531f940be42f1424c3fb44.jpg
C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iris-setosa\iri
```

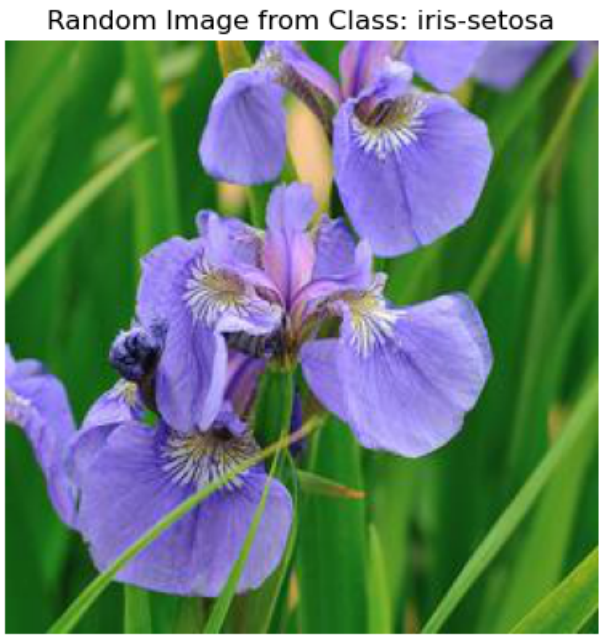
In [3]:
```
data_dir = r"C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset" # Data
data_dir = pathlib.Path(data_dir)
data_dir
```

Out[3]:
```
WindowsPath('C:/Users/Tanmayee/OneDrive/Documents/Personal/Github/Flower Detection/Dataset')
```

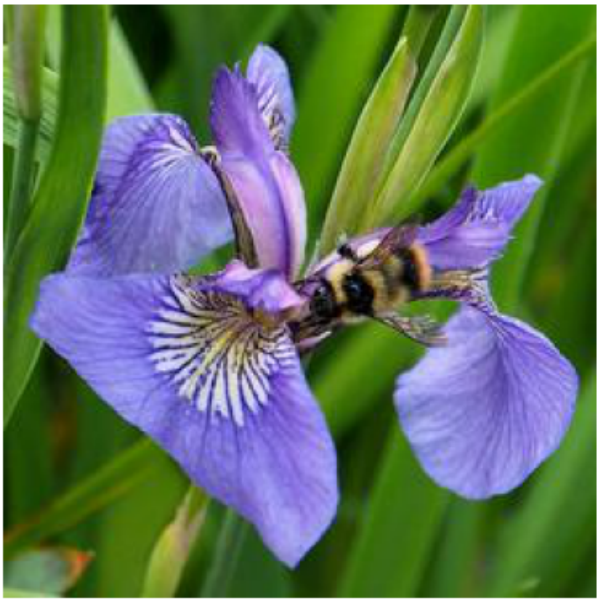
In [4]:
```
# Define class directories
class_directories = ['iris-setosa', 'iris-versicolour', 'iris-virginica']

# Initialize dictionaries to store counts for each class
class_counts = {}

# Count the number of files in each class directory
for class_dir in class_directories:
    class_path = os.path.join(data_dir, class_dir)
    class_count = len(os.listdir(class_path))
    class_counts[class_dir] = class_count

# Print the counts for each class
for class_dir, count in class_counts.items():
    print(f"Length of {class_dir}: {count}")
```

```
Length of iris-setosa: 67
Length of iris-versicolour: 269
Length of iris-virginica: 85
```

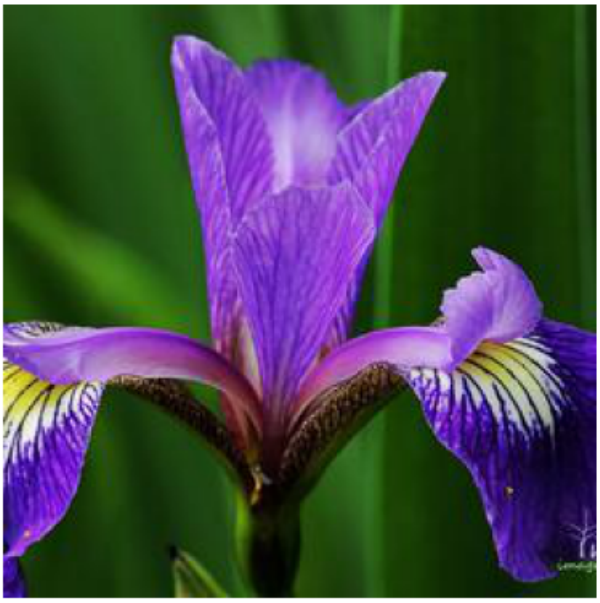
In [5]:
```
# List all subdirectories (each representing a class)
class_directories = [d for d in os.listdir(data_dir) if os.path.isdir(os.path.join(data_dir, d))]

# Number of random images to display
num_images_to_display = 5

# Iterate to display random images
for _ in range(num_images_to_display):
    # Randomly select a class directory
    selected_class = random.choice(class_directories)

    # Get a list of image file paths in the selected class directory
    class_dir_path = os.path.join(data_dir, selected_class)
    image_paths = [os.path.join(class_dir_path, fname) for fname in os.listdir(class_dir_path)]

    # Randomly select an image from the list
    selected_image_path = random.choice(image_paths)

    # Open and display the selected image
    selected_image = Image.open(selected_image_path)
    plt.figure()
    plt.imshow(selected_image)
    plt.title(f"Random Image from Class: {selected_class}")
    plt.axis('off')

# Show all 5 random images
plt.show()
```



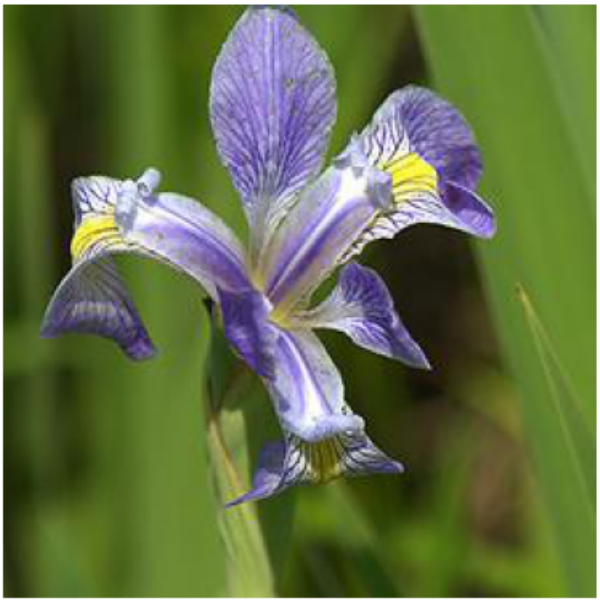
Random Image from Class: iris-setosa



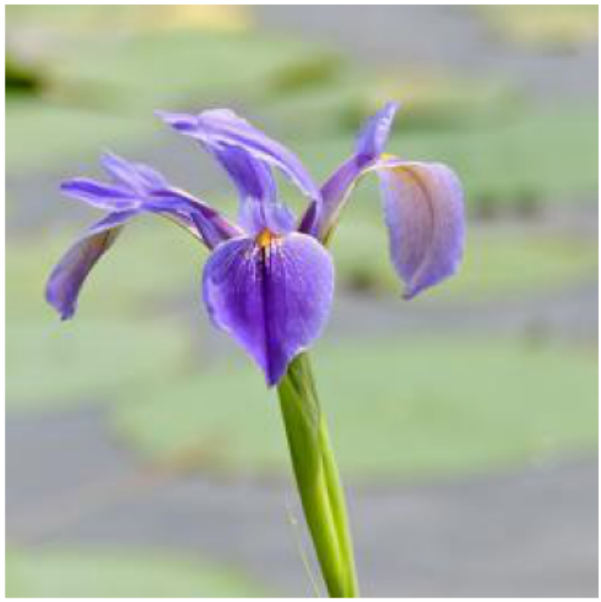
Random Image from Class: iris-versicolour



Random Image from Class: iris-versicolour



Random Image from Class: iris-virginica



In [6]:
```
# contains the images path
df_images = {
    'setosa' : list(data_dir.glob('iris-setosa/*')),
    'versicolour' : list(data_dir.glob('iris-versicolour/*')),
    'virginica': list(data_dir.glob('iris-virginica/*'))
}

# create numerical labels for the categories
class_labels = {
    'setosa' : 0,
    'versicolour' : 1,
    'virginica': 2
}

# Reverse the dictionary to get labels for each image path
df_labels = {image_path: label for label, image_paths in df_images.items() for image_path in image

# Example: Print labels for each image
for image_path, label in df_labels.items():
    print(f"Image Path: {image_path}, Label: {label}")
```

```
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
s-setosa\iris-01ab65973fd487a6cee4c5af1551c42b264eec5abab46bffd7c307ffef647e11.jpg, Label: se
tosa
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
s-setosa\iris-0797945218a97d6e5251b4758a2ba1b418cbd52ce4ef46a3239e4b939bd9807b.jpg, Label: se
tosa
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
s-setosa\iris-0c826b6f4648edf507e0cafdab53712bb6fd1f04dab453cee8db774a728dd640.jpg, Label: se
tosa
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
s-setosa\iris-0ff5ba898a0ec179a25ca217af45374fdd06d606bb85fc29294291facad1776a.jpg, Label: se
tosa
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
s-setosa\iris-1289c57b571e8e98e4feb3e18a890130adc145b971b7e208a6ce5bad945b4a5a.jpg, Label: se
tosa
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
s-setosa\iris-16f7515e1d6aa6d7dd3af4bca38c8065bfab9d426c5fd75b3c4bc51d737fb9d0.jpg, Label: se
tosa
Image Path: C:\Users\Tanmayee\OneDrive\Documents\Personal\Github\Flower Detection\Dataset\iri
```

In [7]:
```
# Get a random index for the 'virginica' category
rand_virginica = random.randint(0, len(df_images['virginica']) - 1)

# Load the random image
random_virginica_image_path = df_images['virginica'][rand_virginica]
img = cv2.imread(str(random_virginica_image_path))

# Check the shape of the image
image_shape = img.shape
print(f"Image Shape: {image_shape}")
```

```
Image Shape: (256, 256, 3)
```

In [8]:
```
from skimage import io
from skimage.color import rgb2gray
from skimage.transform import resize
```

In [9]:
```
# Example labels matching the labels in df_images
df_labels = {
    'setosa' : 0,
    'versicolour' : 1,
    'virginica': 2
}

# Initialize empty lists for images and labels
X, y = [], []

# Iterate through label-image pairs in df_images
for label, images in df_images.items():
    # Load and resize each image
    resized_images = [cv2.resize(cv2.imread(str(image)), (224, 224)) for image in images]

    # Extend X and y with the resized images and corresponding labels
    X.extend(resized_images)
    y.extend([df_labels[label]] * len(resized_images))

# Convert the lists to NumPy arrays
X = np.array(X)
y = np.array(y)

# Check the number of images and labels
print("Number of images:", len(X))
print("Number of labels:", len(y))
```

```
Number of images: 421
Number of labels: 421
```

In [10]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score
```

In [11]:
```
# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [12]:
```
# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)

# Create PyTorch DataLoader for training data
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

# Define a simple feedforward neural network for classification
class Classifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Classifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Define the model, loss function, and optimizer
input_size = X_train.shape[1] * X_train.shape[2] * X_train.shape[3]  # Flatten input
hidden_size = 128
num_classes = len(set(y_train))
model = Classifier(input_size, hidden_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 10
for epoch in range(num_epochs):
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Evaluate the model on the testing data
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
model.eval()
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, predicted = torch.max(outputs, 1)

test_accuracy = accuracy_score(y_test, predicted.numpy())
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
Test Accuracy: 0.5647
```

## Feed Forward Neural Network

In [13]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score
```

In [14]:
```
# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)

# Create PyTorch DataLoader for training data
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

# Calculate input_size based on your input data shape
input_size = X_train_tensor.size(1) * X_train_tensor.size(2) * X_train_tensor.size(3)
```

In [15]:
```
# Define a simple feedforward neural network for classification
class Classifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Classifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Define the model, loss function, and optimizer
hidden_size = 128
num_classes = len(set(y_train))
model = Classifier(input_size, hidden_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 10
for epoch in range(num_epochs):
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Evaluate the model on the testing data
model.eval()
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, predicted = torch.max(outputs, 1)

test_accuracy = accuracy_score(y_test, predicted.numpy())
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
Test Accuracy: 0.5882
```

In [ ]: