Slip 13

13.1

// Volt class to measure volts

class Volt {

private int volts;

public Volt(int volts) {

this.volts = volts;

}

public int getVolts() {

return volts;

}

}

// Socket class producing constant volts of 120V

class Socket {

public Volt getVolt() {

return new Volt(120);

}

}

// Adapter interface

interface SocketAdapter {

Volt get3Volts();

Volt get12Volts();

Volt getDefaultVolts();

}

// Class Adapter implementing the SocketAdapter interface

class SocketClassAdapter extends Socket implements SocketAdapter {

@Override

public Volt get3Volts() {

Volt volt = getVolt();

return new Volt(volt.getVolts() / 40); // Dividing by 40 to get 3 volts

}

@Override

public Volt get12Volts() {

Volt volt = getVolt();

return new Volt(volt.getVolts() / 10); // Dividing by 10 to get 12 volts

}

@Override

public Volt getDefaultVolts() {

return getVolt();

}

}

// Client code to test the Adapter pattern

public class AdapterPatternExample {

public static void main(String[] args) {

SocketAdapter socketAdapter = new SocketClassAdapter();

Volt volt3 = socketAdapter.get3Volts();

System.out.println("3 Volts: " + volt3.getVolts());

Volt volt12 = socketAdapter.get12Volts();

System.out.println("12 Volts: " + volt12.getVolts());

Volt defaultVolt = socketAdapter.getDefaultVolts();

System.out.println("Default Volts: " + defaultVolt.getVolts());

}

}

13.2

# Write a python Program to prepare scatter plot for iris dataset

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

iris = pd.read\_csv("iris.csv")

print(iris.head(20))

plt.plot(iris.Id,iris["sepal.length"],"r--")

plt.show

iris.plot(kind = "scatter", x='sepal.length', y ='petal.length')

plt.show()

13.3

const express = require('express');

const mongoose = require('mongoose');

const bcrypt = require('bcrypt');

const session = require('express-session');

const MongoStore = require('connect-mongo')(session);

const app = express();

const PORT = 3000;

// Connect to MongoDB

mongoose.connect('mongodb://localhost/user\_login\_system', {

useNewUrlParser: true,

useUnifiedTopology: true,

});

// Define User schema

const userSchema = new mongoose.Schema({

username: String,

password: String,

});

const User = mongoose.model('User', userSchema);

// Middleware

app.use(express.urlencoded({ extended: true }));

app.use(

session({

secret: 'your-secret-key',

resave: false,

saveUninitialized: true,

store: new MongoStore({ mongooseConnection: mongoose.connection }),

})

);

// Routes

app.get('/', (req, res) => {

res.send('Welcome to the User Login System');

});

app.get('/login', (req, res) => {

res.send('Login Page');

});

app.post('/login', async (req, res) => {

const { username, password } = req.body;

try {

const user = await User.findOne({ username });

if (user && bcrypt.compareSync(password, user.password)) {

req.session.user = user;

res.redirect('/dashboard');

} else {

res.send('Invalid username or password');

}

} catch (error) {

res.status(500).send('Internal Server Error');

}

});

app.get('/dashboard', (req, res) => {

if (req.session.user) {

res.send(`Welcome ${req.session.user.username} to the Dashboard`);

} else {

res.redirect('/login');

}

});

// Server

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});