Problem Statement 1: Create a simple html form with JavaScript validation for fields like email, phone number and password length. Prevent empty or invalid inputs.

```
<!DOCTYPE html>
<html>
<head>
 <title>Simple Form Validation</title>
 <style>
   /* Page setup */
   body {
     font-family: Arial, sans-serif;
     background-color: #e3f2fd; /* light blue background */
     display: flex;
     justify-content: center;
     align-items: center;
     height: 100vh;
     margin: 0;
     animation: fadeIn 1s ease-in;
   }

   /* Form box */
   form {
     background: white;
     padding: 30px 30px; /* equal padding left & right */
     border-radius: 12px;
     box-shadow: 0 4px 15px rgba(0,0,0,0.1);
     width: 320px;
     opacity: 0;
     animation: slideUp 1s forwards;
   }

   /* Animations */
   @keyframes fadeIn {
     from { opacity: 0; }
     to { opacity: 1; }
   }
   @keyframes slideUp {
     from { transform: translateY(30px); opacity: 0; }
     to { transform: translateY(0); opacity: 1; }
   }
```

```
/* Heading */
h2 {
  text-align: center;
  color: #1565c0;
  margin-top: 0;
}

/* Labels and inputs */
label {
  display: block;
  margin-top: 12px;
  font-weight: bold;
  color: #0d47a1;
}

input {
  width: 100%;
  padding: 8px;
  margin-top: 6px;
  border: 1px solid #90caf9;
  border-radius: 5px;
  transition: 0.3s;
  box-sizing: border-box;
}

input:focus {
  border-color: #1976d2;
  box-shadow: 0 0 8px rgba(25,118,210,0.4);
  outline: none;
}

/* Button */
button {
  margin-top: 18px;
  width: 100%;
  padding: 10px;
  background-color: #1976d2;
  color: white;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  font-size: 15px;
  transition: 0.3s;
}
```

```
    button:hover {
      background-color: #0d47a1;
      transform: scale(1.03);
    }

    /* Error message */
    .error {
      color: red;
      font-size: 13px;
    }

    /* Success message tab */
    .success {
      margin-top: 15px;
      text-align: center;
      background-color: #bbdefb;
      color: #0d47a1;
      border-radius: 5px;
      padding: 10px;
      display: none; /* hidden until success */
      animation: fadeIn 0.8s;
      font-weight: bold;
    }
  </style>
</head>
<body>

  <form id="myForm" onsubmit="return validateForm()">
    <h2>Registration Form</h2>

    <label>Email:</label>
    <input type="text" id="email">
    <span id="emailError" class="error"></span>

    <label>Phone Number:</label>
    <input type="text" id="phone">
    <span id="phoneError" class="error"></span>

    <label>Password:</label>
    <input type="password" id="password">
    <span id="passwordError" class="error"></span>

    <button type="submit">Submit</button>
  <div id="successTab" class="success">Form submitted successfully!</div>
</form>

<script>
  function validateForm() {
    var email = document.getElementById("email").value.trim();
    var phone = document.getElementById("phone").value.trim();
    var password = document.getElementById("password").value.trim();

    var emailError = document.getElementById("emailError");
    var phoneError = document.getElementById("phoneError");
    var passwordError = document.getElementById("passwordError");
    var successTab = document.getElementById("successTab");

    // Clear old errors and hide success tab
    emailError.textContent = "";
    phoneError.textContent = "";
    passwordError.textContent = "";
    successTab.style.display = "none";

    var emailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;
    var phonePattern = /^[0-9]{10}$/;
    var valid = true;

    if (email === "") {
      emailError.textContent = "Email is required";
      valid = false;
    } else if (!email.match(emailPattern)) {
      emailError.textContent = "Enter a valid email";
      valid = false;
    }

    if (phone === "") {
      phoneError.textContent = "Phone number is required";
      valid = false;
    } else if (!phone.match(phonePattern)) {
      phoneError.textContent = "Enter 10-digit phone number";
      valid = false;
    }

    if (password === "") {
      passwordError.textContent = "Password is required";
      valid = false;
      } else if (password.length < 6) {
        passwordError.textContent = "Password must be at least 6 characters";
        valid = false;
      }

      if (valid) {
        successTab.style.display = "block"; // Show success tab
      }

      return false; // stop actual form submission
    }
  </script>

</body>
</html>
```

Problem Statement 2: Demonstrate a SQL Injection vulnerability using a basic login form and then fix it by using prepared statements.

CREATE A MAIN FOLDER THAT WILL INCLUDE :

- Subfolder - templates - login.html
- File - init_db , vulnerable_app, fixed_app

Commands -

- cd
- python --version
- python -m pip install --upgrade pip
- python -m pip install flask
- python init_db.py
- python vulnerable_app.py

Open your browser: http://127.0.0.1:5000
Normal credentials:

    - username: alice
    - password: wonderland
    - username: ' OR '1'='1' --
    - password: x

- python fixed_app.py

login.html :

```
<!doctype html>
<html>
 <head><meta charset="utf-8"><title>Login</title></head>
 <body>
   <br><br>
   <center>
     <h2>Login</h2>

     <form method="post" action="/login">
       <label>&emsp;Username: <input name="username"></label><br><br>
       <label>&emsp;Password: <input name="password" type="password"></label><br><br>
       <button type="submit">Login</button>
     </form>

     <br>
     {% if message %}
       <p><strong>{{ message }}</strong></p>
     {% endif %}
   </center>
 </body>
</html>
```

vulnerable_app :

```
# vulnerable_app.py
from flask import Flask, request, render_template
import sqlite3

app = Flask(__name__)

def query_db_raw(username, password):
   conn = sqlite3.connect('users.db')
   c = conn.cursor()
   sql = f"SELECT id, username FROM users WHERE username = '{username}' AND password =
'{password}'"
   print("DEBUG SQL:", sql)
   try:
       c.execute(sql)
       row = c.fetchone()
   finally:
       conn.close()
   return row

@app.route('/', methods=['GET'])
def index():
   return render_template('login.html')

@app.route('/login', methods=['POST'])
```

```
def login():
    username = request.form.get('username', '')
    password = request.form.get('password', '')
    user = query_db_raw(username, password)
    if user:
        return render_template('login.html', message=f"Logged in as {user[1]}")
    else:
        return render_template('login.html', message="Login failed")

if __name__ == '__main__':
    app.run(debug=True)
```

fixed_app.py

```
from flask import Flask, request, render_template
import sqlite3

app = Flask(__name__)

def query_db_prepared(username, password):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    sql = "SELECT id, username FROM users WHERE username = ? AND password = ?"
    print("DEBUG (prepared):", sql, "params:", (username, password))
    try:
        c.execute(sql, (username, password))
        row = c.fetchone()
    finally:
        conn.close()
    return row

@app.route('/', methods=['GET'])
def index():
    return render_template('login.html')

@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username', '')
    password = request.form.get('password', '')
    user = query_db_prepared(username, password)
    if user:
        return render_template('login.html', message=f"Logged in as {user[1]}")
    else:
        return render_template('login.html', message="Login failed")
```

```
if __name__ == '__main__':
   app.run(debug=True)
```

Init_db.py

```
import sqlite3

conn = sqlite3.connect('users.db')
c = conn.cursor()

c.execute('''
CREATE TABLE IF NOT EXISTS users (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   username TEXT UNIQUE,
   password TEXT
)
''')

try:
   c.execute("INSERT INTO users (username, password) VALUES (?, ?)",
          ("alice", "wonderland"))
   conn.commit()
   print("Inserted user alice / wonderland")
except Exception as e:
   print("User may already exist:", e)

conn.close()
```

Problem Statement 3: Use Git to create a repository, make commits and push code to GitHub. Show how version control maintains code integrity.

CMD

- git --version
- git config --global user.name "Vanshita"
- git config --global user.email "vanshita.parab17627@sakec.ac.in"
- mkdir repo
- cd repo
- git init
- echo "print('Hello')" > hello.py
- git status
- git add hello.py
- git commit -m "Added hello.py with print statement"

GITHUB

- Create and Add a new repository
- Copy https://github.com/purva-mahamunkar/first-repo.git

CMD

- git remote add origin https://github.com/Vanshita-Parab/repo.git
- git branch -M main
- git push -u origin main
- echo print("Hello, How are you?") > hello.py
- git add hello.py
- git commit -m "Updated greeting message"
- git push
- git log --oneline
- git cat-file -p HEAD

Problem Statement 4: Open Burp Suite, intercept a request from a demo site and display captured HTTP headers. Identify any insecure parameter.

- Install Burp Suite from Port Swigger for community
- Launch the Burp Suite
- Click on Proxy
- Click on Intercept on
- Open Browser
- Search for 127.0.0.1:8080
- Download CA Certificate
- Minimize the screen
- Check the Request column
- Go the minimized screen
- Search OWASP Juice Shop
- Go to the main screen
- Click on the second line
- Click on Forward
- Click on the second option

Problem Statement 6: Write a small Python or Javascript program to encrypt and decrypt a message using a simple Caesar cipher technique.

```
def caesar_cipher(text, shift, mode):
  result = ""
  for char in text:
      if char.isalpha():  # Encrypt or decrypt letters only
          shift_base = 65 if char.isupper() else 97
          if mode == 'encrypt':
                result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
            elif mode == 'decrypt':
                result += chr((ord(char) - shift_base - shift) % 26 + shift_base)
        else:
            result += char
    return result

print("=== Caesar Cipher Program ===")
choice = input("Do you want to Encrypt or Decrypt a message? (E/D): ").strip().lower()
message = input("Enter your message: ")
shift_value = int(input("Enter shift value (e.g. 3): "))

if choice == 'e':
    encrypted = caesar_cipher(message, shift_value, 'encrypt')
    print("\nEncrypted Message:", encrypted)
elif choice == 'd':
    decrypted = caesar_cipher(message, shift_value, 'decrypt')
    print("\nDecrypted Message:", decrypted)
else:
    print("\nInvalid choice! Please enter 'E' or 'D'.")
```

Problem Statement 7: Build a basic login form that sets and clears a session cookie when the user logs in or logs out.

Create a folder and put 5 files into it

1. user.php
2. index.php
3. authenticate.php
4. dashboard.php

Install php and set environment variables is not installed

Run the following commands :

- php -v
- cd
- php -S localhost:8000

http://localhost:8000

Username : bob
Password : secret

users.php

```
<?php
// users.php
// Demo user store. In real apps use a database.
return [
   // username => password_hash('password123', PASSWORD_DEFAULT)
   'alice' => '$2y$10$9aP5w1y1qXbQZ0s9Gf0vEu5l7wN9UoJvH2zQ0q9Hnqg5KQZ7H3bW6',
   'bob'   => password_hash('secret', PASSWORD_DEFAULT), // generate at runtime
];
?>
```

index.php

```
<?php
// index.php
session_start();

// If already logged in, send to dashboard
if (!empty($_SESSION['user'])) {
   header('Location: dashboard.php');
   exit;
}

// Show any message (e.g. after logout)
$msg = $_GET['msg'] ?? '';
?>
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Login</title></head>
<body>
 <h2>Login</h2>
 <?php if ($msg): ?>
   <p style="color:green;"><?php echo htmlspecialchars($msg); ?></p>
 <?php endif; ?>

 <form method="post" action="authenticate.php">
   <label>Username:<br>
     <input name="username" required autofocus>
   </label><br><br>

   <label>Password:<br>
     <input name="password" type="password" required>
   </label><br><br>

   <button type="submit">Login</button>
 </form>
```

```
</body>
</html>
```

authenticate.php

```php
<?php
// authenticate.php
declare(strict_types=1);

// Configure session cookie params before session_start
$cookieParams = session_get_cookie_params();
session_set_cookie_params([
  'lifetime' => 0,
  'path' => $cookieParams['path'],
  'domain' => $cookieParams['domain'],
  'httponly' => true,
  'samesite' => 'Lax'
]);

session_start();

// Load users
$users = require __DIR__ . '/users.php';

// Get input
$username = trim($_POST['username'] ?? '');
$password = $_POST['password'] ?? '';

if (!$username || !$password) {
  header('Location: index.php?msg=' . urlencode('Missing username or password'));
  exit;
}

// Verify user
if (isset($users[$username]) && password_verify($password, $users[$username])) {
  session_regenerate_id(true); // prevent fixation
  $_SESSION['user'] = [
     'username' => $username,
     'login_time' => time()
  ];
  header('Location: dashboard.php');
  exit;
} else {
  header('Location: index.php?msg=' . urlencode('Invalid credentials'));
  exit;
}
?>
```

dashboard.php

```
<?php
// dashboard.php
session_start();

if (empty($_SESSION['user'])) {
   header('Location: index.php?msg=' . urlencode('Please login first'));
   exit;
}

$user = $_SESSION['user'];
?>
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Dashboard</title></head>
<body>
 <h2>Welcome, <?php echo htmlspecialchars($user['username']); ?></h2>
 <p>Logged in at: <?php echo date('Y-m-d H:i:s', $user['login_time']); ?></p>

 <form method="post" action="logout.php">
   <button type="submit">Logout</button>
 </form>
</body>
</html>
```

logout.php

```
<?php
// logout.php
session_start();

// Clear all session variables
$_SESSION = [];

// Destroy the session and cookie
if (ini_get("session.use_cookies")) {
   $params = session_get_cookie_params();
   setcookie(session_name(), '', time() - 42000,
       $params["path"], $params["domain"],
       $params["secure"], $params["httponly"]
   );
}
```

```
session_destroy();

header('Location: index.php?msg=' . urlencode('You have been logged out successfully'));
exit;
?>
```

Problem Statement 8: Create a feedback form with both frontend and backend validation(e.g. prevent script tags or blank submissions)

Feedback Form (Frontend + Backend Validation)

1. Create project folder:
   - mkdir feedback_form
   - cd feedback_form

2. Create files:
   - index.html (frontend form + JS validation)
   - server.js (Express backend validation)

3. Initialize & install:
   - npm init -y
   - npm install express

4. Start server:
   - node server.js
   - Open: http://localhost:3000

5. Quick tests:
   - Valid input: John Doe, john@example.com, rating 1–5, feedback ≥10 chars → success.
   - Invalid inputs to check:
   - Blank fields → rejected
   - Name with numbers/symbols → rejected
   - Feedback with <script> → rejected
   - Feedback containing select|drop|--|; → rejected

6. Notes:
   - Frontend uses HTML required, type="email", min/max and JS checks.
   - Backend re-validates, strips/blocks <script>, escapes output before echoing.
   - This is a demo — for production use robust sanitizers, HTTPS, and parameterized DB queries.

index.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
 <meta charset="UTF-8">
 <title>Feedback Form</title>
 <style>
   body {
     font-family: Arial, sans-serif;
     background: linear-gradient(135deg, #066ebd, #4dcbdf);
     color: #333;
     padding: 20px;
   }
   form {
     background: white;
     border-radius: 15px;
     padding: 20px;
     width: 350px;
     margin: auto;
     box-shadow: 0 4px 10px rgba(0,0,0,0.1);
   }
   input, textarea {
     width: 100%;
     padding: 10px;
     margin: 8px 0;
     border-radius: 5px;
     border: 1px solid #ccc;
     box-sizing: border-box;
   }
   button {
     width: 100%;
     padding: 10px;
     background-color: #5a90cd;
     border: none;
     color: white;
     border-radius: 5px;
     font-size: 16px;
     cursor: pointer;
   }
   button:disabled {
     background-color: #ccc;
     cursor: not-allowed;
   }
   .error { color: red; font-size: 13px; }
   .success { color: green; font-size: 14px; margin-top: 10px; }
 </style>
</head>
```

```
<body>
 <h2 align="center"> Feedback Form </h2>

 <form id="feedbackForm" action="http://localhost:3000/submit" method="POST" onsubmit="return
validateForm()">
   <label>Name:</label>
   <input type="text" id="name" name="name" required><br>

   <label>Email:</label>
   <input type="email" id="email" name="email" required><br>

   <label>Rating (1–5):</label>
   <input type="number" id="rating" name="rating" min="1" max="5" required><br>

   <label>Feedback Message:</label>
   <textarea id="feedback" name="feedback" rows="4" minlength="10" required></textarea><br>

   <button type="submit" id="submitBtn">Submit</button>
   <div id="msg" class=""></div>
 </form>

 <script>
   // Disable submit if fields are invalid
   const form = document.getElementById("feedbackForm");
   const submitBtn = document.getElementById("submitBtn");
   const msgDiv = document.getElementById("msg");

   form.addEventListener("input", () => {
     submitBtn.disabled = !form.checkValidity();
     msgDiv.textContent = "";
     msgDiv.className = "";
   });

   // Frontend validation
   function validateForm() {
     const name = document.getElementById("name").value.trim();
     const rating = parseInt(document.getElementById("rating").value);
     const feedback = document.getElementById("feedback").value.trim();

     // Prevent simple script tags
     const scriptTagPattern = /<\s*script\b/i;
     if (scriptTagPattern.test(name) || scriptTagPattern.test(feedback)) {
       showMessage(" Script tags are not allowed!", "error");
       return false;
```

```
    }

    // Name validation (letters, spaces only)
    if (!/^[A-Za-z\s]+$/.test(name)) {
      showMessage(" Name must contain only letters and spaces!", "error");
      return false;
    }

    // Rating validation
    if (!Number.isInteger(rating) || rating < 1 || rating > 5) {
      showMessage(" Rating must be an integer between 1 and 5!", "error");
      return false;
    }

    // Feedback length
    if (feedback.length < 10) {
      showMessage(" Feedback must be at least 10 characters long!", "error");
      return false;
    }

    // allow normal form submit to server
    return true;
  }

  function showMessage(text, type) {
    msgDiv.textContent = text;
    msgDiv.className = (type === "error") ? "error" : "success";
  }
 </script>
</body>
</html>
```

server.js

```
// server.js
const express = require("express");
const path = require("path");
const app = express();

// parse application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));

// Serve index.html at root
app.get("/", (req, res) => {
 res.sendFile(path.join(__dirname, "index.html"));
```

```
});

app.post("/submit", (req, res) => {
 const { name = "", email = "", rating = "", feedback = "" } = req.body;

 // Trim inputs
 const tName = String(name).trim();
 const tEmail = String(email).trim();
 const tRating = Number(rating);
 const tFeedback = String(feedback).trim();

 // === Backend validation ===

 // Required fields
 if (!tName || !tEmail || !tRating || !tFeedback) {
   return res.send(" Please fill all fields!");
 }

 // Prevent script tags (basic)
 const scriptRegex = /<\s*script\b.*?>.*?<\s*\/\s*script\s*>/i;
 if (scriptRegex.test(tName) || scriptRegex.test(tFeedback)) {
   return res.send(" Script tags are not allowed!");
 }

 // Name validation
 if (!/^[A-Za-z\s]+$/.test(tName)) {
   return res.send(" Invalid name (letters and spaces only).");
 }

 // Email format basic check
 if (!/^[^\s@]+@[^\s@]+\.[^\s@]{2,}$/.test(tEmail)) {
   return res.send(" Invalid email format.");
 }

 // Rating range
 if (!Number.isInteger(tRating) || tRating < 1 || tRating > 5) {
   return res.send(" Rating must be an integer between 1 and 5.");
 }

 // Feedback length
 if (tFeedback.length < 10) {
   return res.send(" Feedback is too short.");
 }
```

```
// Simple forbidden-words check to prevent obvious injection payloads
if (/select|drop|insert|delete|update|--|;|\/\*/i.test(tFeedback)) {
  return res.send(" Feedback contains forbidden words or characters.");
}

// Basic HTML-escape before echoing back (avoid reflected XSS)
function escapeHtml(s) {
  return s
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/"/g, "&quot;")
    .replace(/'/g, "&#039;");
}

// Success response (show escaped values)
return res.send(`
   Feedback submitted successfully!<br>
  <b>Name:</b> ${escapeHtml(tName)}<br>
  <b>Email:</b> ${escapeHtml(tEmail)}<br>
  <b>Rating:</b> ${escapeHtml(String(tRating))}<br>
  <b>Feedback:</b> ${escapeHtml(tFeedback)}<br>
  <br><a href="/">Submit another</a>
`);
});

app.listen(3000, () => console.log("🚀 Server running at http://localhost:3000"));
```

Input Field	Frontend Validation Example	Backend Validation Example
Name	Must not be empty (use required in HTML or JS check)	Check if name only contains letters (no numbers/symbols)
Email	Must be a valid email format (use type="email")	Check if email already exists in database
Rating	Must be between 1–5 (HTML min, max attributes or JS check)	Validate rating range again on server side
Feedback Message	Must be at least 10 characters (JS check)	Check for SQL injection or restricted words before saving
Submit Button	Disabled if fields are invalid (JS)	Reject invalid data on server

Problem Statement 9: Implement a password strength checker using JS that validates minimum length, use of numbers and special characters

Create a folder

Make just one file in it - index.html

Write the following code in it :

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Password Strength Checker</title>
 <style>
   body {
     font-family: Arial, sans-serif;
     background: linear-gradient(to right, #89f7fe, #66a6ff);
     display: flex;
     flex-direction: column;
     align-items: center;
     justify-content: center;
     height: 100vh;
     color: #333;
   }

   .password-box{
     align-items: center;
     display: flex;
     flex-direction: column;
     margin-top: 10px;
   }

   .show-password input[type="checkbox"] {
       cursor: pointer;
       accent-color: #004aad;
   }

   #togglePassword {
       cursor: pointer;
       accent-color: #004aad;
   }
```

```
h2 {
  color: #004aad;
  margin-bottom: 20px;
}

input {
  padding: 10px;
  width: 260px;
  border: 2px solid #ccc;
  border-radius: 8px;
  font-size: 16px;
  outline: none;
  transition: border-color 0.3s;
}

#strengthBar {
  width: 260px;
  height: 10px;
  margin-top: 10px;
  border-radius: 5px;
  background-color: #ddd;
  overflow: hidden;
}

#strengthFill {
  height: 100%;
  width: 0%;
  background-color: red;
  transition: width 0.4s, background-color 0.4s;
}

#strengthMessage {
  margin-top: 12px;
  font-weight: bold;
}

ul {
  text-align: left;
  list-style-type: none;
  margin-top: 15px;
  padding: 0;
}

li {
      font-size: 14px;
      margin: 3px 0;
    }

    .valid {
      color: green;
    }

    .invalid {
      color: red;
    }
  </style>
</head>
<body>
  <h2>Password Strength Checker</h2>
  <div class="password-box">
    <input type="password" id="password" placeholder="Enter your password">

    <div class="show-password">
        <input type="checkbox" id="togglePassword">
    </div>
  </div>
   <div id="strengthBar"><div id="strengthFill"></div></div>
  <div id="strengthMessage"></div>

  <ul>
    <li id="length" class="invalid">Minimum 8 characters</li>
    <li id="number" class="invalid">Must contain at least one number</li>
    <li id="special" class="invalid">Must contain at least one special character</li>
  </ul>

  <script>
    const password = document.getElementById('password');
    const fill = document.getElementById('strengthFill');
    const message = document.getElementById('strengthMessage');

    const lengthReq = document.getElementById('length');
    const numberReq = document.getElementById('number');
    const specialReq = document.getElementById('special');

    const togglePassword = document.getElementById('togglePassword');

    togglePassword.addEventListener('change', () => {
        password.type = togglePassword.checked ? 'text' : 'password';
    });

    password.addEventListener('input', () => {
      const value = password.value;
      const hasNumber = /\d/.test(value);
      const hasSpecial = /[!@#$%^&*(),.?":{}|<>]/.test(value);
      const minLength = value.length >= 8;

      // Update checklist
      lengthReq.className = minLength ? 'valid' : 'invalid';
      lengthReq.textContent = minLength ? 'Minimum 8 characters (OK)' : 'Minimum 8 characters';

      numberReq.className = hasNumber ? 'valid' : 'invalid';
      numberReq.textContent = hasNumber ? 'Contains at least one number (OK)' : 'Must contain at least
one number';

      specialReq.className = hasSpecial ? 'valid' : 'invalid';
      specialReq.textContent = hasSpecial ? 'Contains at least one special character (OK)' : 'Must contain at
least one special character';

      // Determine strength
      let strength = 0;
      if (minLength) strength++;
      if (hasNumber) strength++;
      if (hasSpecial) strength++;

      // Update progress bar and text
      if (strength === 0) {
        fill.style.width = '0%';
        fill.style.backgroundColor = 'red';
        message.textContent = '';
      } else if (strength === 1) {
        fill.style.width = '33%';
        fill.style.backgroundColor = 'red';
        message.textContent = 'Weak password';
      } else if (strength === 2) {
        fill.style.width = '66%';
        fill.style.backgroundColor = 'orange';
        message.textContent = 'Medium strength password';
      } else if (strength === 3) {
        fill.style.width = '100%';
        fill.style.backgroundColor = 'green';
        message.textContent = 'Strong password';
      }
```

```
   });
 </script>
</body>
</html>
```

Problem Statement 10: Demonstrate how to generate and use a personal access token in GitHub instead of a password for secure authentication.

- Log in to https://github.com
- Click your profile picture → Settings
- On the left sidebar, click Developer settings
- Then click Personal access tokens → Tokens (classic)
- Click Generate new token (classic)
- Note: Git Access Token for Purva
- Expiration: e.g., 90 days or “No expiration”
- Select Scopes (Permissions) — tick:
    - repo → full control of private and public repositories
    - workflow → if using GitHub Actions
    - (Optional) user if you want to access profile info
- Scroll down and click Generate token
- Copy the generate PAT
- mkdir repo
- cd repo
- git remote -v
- git status
- git push origin main
- A github browser will open and paste the PAT