2. Implement the Calculator using a switch case in TypeScript.

calc.ts:

function calculate() {

const num1Input = document.getElementById('num1') as HTMLInputElement;

const num2Input = document.getElementById('num2') as HTMLInputElement;

const operatorSelect = document.getElementById('operator') as HTMLSelectElement;

const resultParagraph = document.getElementById('result');

const num1 = parseFloat(num1Input.value);

const num2 = parseFloat(num2Input.value);

const operator = operatorSelect.value;

let result: number;

switch (operator) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

if (num2 !== 0) {

result = num1 / num2;

} else {

result = NaN; // Indicate division by zero

}

break;

default:

result = NaN; // Indicate invalid operator

break;

}

resultParagraph.textContent = isNaN(result) ? "Invalid input" : `Result: ${result}`;

}

Calc.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Calculator</title>

</head>

<body>

<h2>Simple Calculator</h2>

<input type="number" id="num1">

<select id="operator">

<option value="+">+</option>

<option value="-">-</option>

<option value="\*">\*</option>

<option value="/">/</option>

</select>

<input type="number" id="num2">

<button onclick="calculate()">Calculate</button>

<p id="result">Result: </p>

<script src="calc.js"></script>

</body>

</html>

3. Implement the all types Inheritance using TypeScript

inh.ts:

// Base class for all students

class Student {

constructor(public name: string, public age: number) {}

display(): string {

return `Student: ${this.name}, Age: ${this.age}`;

}

}

// Subclass for undergraduate students, inheriting from Student

class Undergraduate extends Student {

constructor(public name: string, public age: number, public major: string) {

super(name, age);

}

display(): string {

return `Undergraduate Student: ${this.name}, Age: ${this.age}, Major: ${this.major}`;

}

}

// Multilevel Inheritance

class GraduateStudent extends Undergraduate {

enrollInCourse(course: string): string {

return `${this.name} enrolled in the course: ${course}`;

}

}

// Output on the webpage

const outputElement = document.getElementById('output');

// Single Inheritance (Student to Undergraduate)

const undergraduate = new Undergraduate('John Doe', 20, 'Computer Science');

outputElement.innerHTML += `<p>Single Inheritance: ${undergraduate.display()}</p>`;

// Multilevel Inheritance (Undergraduate to GraduateStudent)

const graduateStudent = new GraduateStudent('Jane Smith', 25, 'Physics');

outputElement.innerHTML += `<p>Multilevel Inheritance: ${graduateStudent.display()}, ${graduateStudent.enrollInCourse('Quantum Mechanics')}</p>`;

Inh.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Inheritance Types</title>

<script defer src="inh.js"></script>

</head>

<body>

<h1>Inheritance Types in TypeScript</h1>

<div id="output"></div>

</body>

</html>

4. Implement the all Access Modifiers using TypeScript

acc.ts:

// Public class and members

class Car {

// Public members are accessible from anywhere

public brand: string;

constructor(brand: string) {

this.brand = brand;

}

// Public method

public start(): string {

return `${this.brand} is starting...`;

}

}

// Protected class and members

class Engine extends Car {

// Protected members are accessible within the class and its subclasses

protected horsepower: number;

constructor(brand: string, horsepower: number) {

super(brand);

this.horsepower = horsepower;

}

// Protected method

protected displayHorsepower(): string {

return `${this.brand} has ${this.horsepower} horsepower.`;

}

}

// Private class and members

class CarDetails extends Engine {

// Private members are only accessible within the class

private color: string;

constructor(brand: string, horsepower: number, color: string) {

super(brand, horsepower);

this.color = color;

}

// Public method accessing protected and private members

public displayCarDetails(): string {

return `\n${this.start()}\n ${this.displayHorsepower()}\n Color:

${this.color}`;

}

}

// Creating an instance of CarDetails

const myCar = new CarDetails("Toyota", 200, "Red");

// Accessing public method

const output = myCar.displayCarDetails();

console.log(output);

Acc.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>TypeScript Access Modifiers Example</title>

</head>

<body>

<h1>Car Details</h1>

<div id="carDetails"></div>

<script src="acc.js"></script>

<script>

// Displaying the output on the webpage

const carDetailsDiv = document.getElementById("carDetails");

carDetailsDiv.innerHTML = `

<p>${output}</p>

`;

</script>

</body>

</html>

5. Implement the interface in typescript.

intf.ts:

// Define an interface

interface Person {

name: string;

age: number;

greet(): string;

}

// Implement the interface in a class

class Student implements Person {

constructor(public name: string, public age: number) {}

// Implementing the method defined in the interface

greet(): string {

return `Hello, my name is ${this.name} and I am ${this.age} years old.`;

}

}

// Creating an instance of the Student class

const student = new Student("John", 20);

// Update HTML content with the result

document.getElementById('output').innerHTML += `

<p>${student.greet()}</p>

`;

Intf.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Interface Example</title>

</head>

<body>

<div id="output"></div>

<script src="intf.js"></script>

</body>

</html>

6. Building a Simple Website with TypeScript.

Html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>To-Do List</title>

<link rel="stylesheet" href="to.css">

</head>

<body>

<div class="container">

<h1>To-Do List</h1>

<input type="text" id="taskInput" placeholder="Add a new task...">

<button id="addTaskButton">Add Task</button>

<ul id="taskList"></ul>

</div>

<script src="to.js"></script>

</body>

</html>

Css:

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 500px;

margin: 0 auto;

padding: 20px;

}

h1 {

text-align: center;

}

input[type="text"] {

width: 70%;

padding: 10px;

font-size: 16px;

}

button {

padding: 10px 20px;

background-color: #007bff;

color: #fff;

border: none;

cursor: pointer;

margin-left: 10px;

}

button:hover {

background-color: #0056b3;

}

ul {

list-style-type: none;

padding: 0;

}

li {

margin-bottom: 10px;

}

.completed {

text-decoration: line-through;

color: #888;

}

Ts:

document.addEventListener('DOMContentLoaded', () => {

const taskInput = document.getElementById('taskInput') as HTMLInputElement;

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

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

addTaskButton.addEventListener('click', () => {

const taskText = taskInput.value.trim();

if (taskText !== '') {

addTask(taskText);

taskInput.value = '';

}

});

function addTask(taskText: string) {

const li = document.createElement('li');

li.textContent = taskText;

li.addEventListener('click', () => {

li.classList.toggle('completed');

});

taskList.appendChild(li);

}

});

7. Create an application like Students Record using Angular.

8. Create an application like Feedback System using Angular.

9. Create a simple HTML web page Project using Angular Framework and apply ng-controller, ng-

model and expressions.

10. Implement the Events and Validations in Angular. (Create functions and add events, adding adding

HTML validators)

11. Write a program to use AJAX for user validation using and to show the result on the same page

below the submit button.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Validation</title>

<script>

function validateUser() {

var name = document.getElementById("username").value;

var resultDiv = document.getElementById("result");

if (name.trim().length > 0) {

if (name.trim() === "kjsit") {

resultDiv.textContent = "Welcome, kjsit!";

} else {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

resultDiv.textContent = this.responseText;

}

};

xhttp.open("GET", "dummy\_endpoint.php?name=" + name, true);

xhttp.send();

}

} else {

resultDiv.textContent = "Please enter a valid name!";

}

}

</script>

</head>

<body>

<form>

<label for="username">Enter your name:</label>

<input type="text" id="username" name="username">

<button type="button" onclick="validateUser()">Submit</button>

</form>

<div id="result"></div>

</body>

</html>

12. Implement CRUD operations using MongoDB in different scenarios.

Scenario 1: Basic CRUD Operations

1. Create a Database:

mongo

1. Create a Collection:

use mydb

1. Insert Data:

db.users.insertOne({ name: "John", age: 30 })

1. Read Data:

db.users.find()

1. Update Data:

db.users.updateOne({ name: "John" }, { $set: { age: 35 } })

1. Delete Data:

db.users.deleteOne({ name: "John" })

13. Application using MongoDB and Nodejs.

npm init -y

npm install express mongoose

Node app.js

app.js

const express = require('express');

const mongoose = require('mongoose');

const path = require('path');

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/nodejs\_mongodb\_app', {

useNewUrlParser: true,

useUnifiedTopology: true

});

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'MongoDB connection error:'));

db.once('open', () => {

console.log('Connected to MongoDB');

});

// Create a User model

const User = mongoose.model('User', {

name: String,

email: String,

age: Number

});

const app = express();

app.use(express.json());

app.use(express.static(path.join(\_\_dirname, 'public')));

// Route to serve the index page

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

res.sendFile(path.join(\_\_dirname, 'views', 'index.html'));

});

// Route to create a new user

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

try {

const user = new User(req.body);

await user.save();

res.status(201).send(user);

} catch (err) {

res.status(400).send(err);

}

});

// Route to serve the create page

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

res.sendFile(path.join(\_\_dirname, 'views', 'create.html'));

});

// Route to serve the edit page

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

res.sendFile(path.join(\_\_dirname, 'views', 'edit.html'));

});

// Route to get all users

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

try {

const users = await User.find();

res.send(users);

} catch (err) {

res.status(500).send(err);

}

});

// Route to update a user

app.patch('/users/:id', async (req, res) => {

try {

const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.send(user);

} catch (err) {

res.status(400).send(err);

}

});

// Route to delete a user

app.delete('/users/:id', async (req, res) => {

try {

await User.findByIdAndDelete(req.params.id);

res.send('User deleted successfully');

} catch (err) {

res.status(500).send(err);

}

});

// Start the server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

views/

Index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Management</title>

</head>

<body>

<h1>User Management</h1>

<a href="/create">Create New User</a>

<ul id="users-list"></ul>

<script>

fetch('/users')

.then(response => response.json())

.then(users => {

const userList = document.getElementById('users-list');

userList.innerHTML = users.map(user => `<li>${user.name} - ${user.email} - ${user.age} <a href="/edit/${user.\_id}">Edit</a></li>`).join('');

})

.catch(error => console.error('Error fetching users:', error));

</script>

</body>

</html>

Create.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Create User</title>

</head>

<body>

<h1>Create User</h1>

<form action="/users" method="post">

<label for="name">Name:</label><br>

<input type="text" id="name" name="name" required><br>

<label for="email">Email:</label><br>

<input type="email" id="email" name="email" required><br>

<label for="age">Age:</label><br>

<input type="number" id="age" name="age" required><br>

<button type="submit">Create</button>

</form>

</body>

</html>

Edit.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Edit User</title>

</head>

<body>

<h1>Edit User</h1>

<form id="edit-form">

<label for="name">Name:</label><br>

<input type="text" id="name" name="name" required><br>

<label for="email">Email:</label><br>

<input type="email" id="email" name="email" required><br>

<label for="age">Age:</label><br>

<input type="number" id="age" name="age" required><br>

<button type="submit">Save</button>

</form>

<script>

const userId = window.location.pathname.split('/').pop();

fetch(`/users/${userId}`)

.then(response => response.json())

.then(user => {

document.getElementById('name').value = user.name;

document.getElementById('email').value = user.email;

document.getElementById('age').value = user.age;

})

.catch(error => console.error('Error fetching user:', error));

document.getElementById('edit-form').addEventListener('submit', event => {

event.preventDefault();

const name = document.getElementById('name').value;

const email = document.getElementById('email').value;

const age = document.getElementById('age').value;

fetch(`/users/${userId}`, {

method: 'PATCH',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ name, email, age })

})

.then(response => response.json())

.then(user => {

alert('User updated successfully');

window.location.href = '/';

})

.catch(error => console.error('Error updating user:', error));

});

</script>

</body>

</html>

14. Design Feedback Form using Flask.

index.html:

<!DOCTYPE html>

<html>

<head>

<title>Feedback Form</title>

</head>

<body>

<h2>Feedback Form</h2>

<form action="/submit\_feedback" method="post">

<label for="name">Name:</label><br>

<input type="text" id="name" name="name" required><br>

<label for="email">Email:</label><br>

<input type="email" id="email" name="email" required><br>

<label for="rating">Rating:</label><br>

<select id="rating" name="rating" required>

<option value="1">1 - Poor</option>

<option value="2">2 - Fair</option>

<option value="3">3 - Average</option>

<option value="4">4 - Good</option>

<option value="5">5 - Excellent</option>

</select><br>

<label for="feedback">Feedback:</label><br>

<textarea id="feedback" name="feedback" required></textarea><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

Thank\_you.html:

<!DOCTYPE html>

<html>

<head>

<title>Thank You</title>

</head>

<body>

<h2>Thank You for Your Feedback!</h2>

<p>We appreciate your time and feedback.</p>

<a href="/">Back to Home</a>

</body>

</html>

app.py:

from flask import Flask, render\_template, request, redirect, url\_for

app = Flask(\_\_name\_\_)

# This is just a temporary storage for feedback, in a real application, you'd use a database.

feedback\_data = []

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/submit\_feedback', methods=['POST'])

def submit\_feedback():

name = request.form['name']

email = request.form['email']

rating = request.form['rating']

feedback = request.form['feedback']

# Store the feedback data

feedback\_data.append({

'name': name,

'email': email,

'rating': rating,

'feedback': feedback

})

return redirect(url\_for('thank\_you'))

@app.route('/thank\_you')

def thank\_you():

return render\_template('thank\_you.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

15. Design Weather App using Flask.

Index.html:

<!DOCTYPE html>

<html>

<head>

<title>Weather App</title>

</head>

<body>

<h1>Weather App</h1>

<form action="/weather" method="post">

<label for="city">Enter city:</label>

<input type="text" id="city" name="city" required>

<input type="submit" value="Get Weather">

</form>

</body>

</html>

weather.html:

<!DOCTYPE html>

<html>

<head>

<title>Weather Info</title>

</head>

<body>

<h1>Weather Info for {{ weather\_info.city }}</h1>

{% if weather\_info %}

<p>Temperature: {{ weather\_info.temperature }}°C</p>

<p>Description: {{ weather\_info.description }}</p>

{% else %}

<p>No weather data found for the specified city.</p>

{% endif %}

<a href="/">Back to Home</a>

</body>

</html>

app.py:

import requests

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

def get\_weather(city):

url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid=0ecfd5110bbf55cc9399e1e4ccd15c7e&units=metric'

response = requests.get(url)

data = response.json()

return data

@app.route('/')

def index():

return render\_template('index.html')

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

def weather():

city = request.form['city']

weather\_data = get\_weather(city)

if weather\_data['cod'] == 200:

weather\_info = {

'city': city,

'temperature': weather\_data['main']['temp'],

'description': weather\_data['weather'][0]['description']

}

else:

weather\_info = None

return render\_template('weather.html', weather\_info=weather\_info)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

16. Design Portfolio Website using Flask.

Home.html:

<!DOCTYPE html>

<html>

<head>

<title>Home - Your Name</title>

</head>

<body>

<h1>Welcome to My Portfolio</h1>

<p>This is the home page of your portfolio website.</p>

<p>Feel free to explore!</p>

<a href="/about">About Me</a>

<a href="/portfolio">Portfolio</a>

<a href="/contact">Contact Me</a>

</body>

</html>

About.html

<!DOCTYPE html>

<html>

<head>

<title>About Me - Your Name</title>

</head>

<body>

<h1>About Me</h1>

<p>This is the about page of your portfolio website.</p>

<p>Write something about yourself here.</p>

<a href="/">Home</a>

<a href="/portfolio">Portfolio</a>

<a href="/contact">Contact Me</a>

</body>

</html>

Contact.html

<!DOCTYPE html>

<html>

<head>

<title>Contact Me - Your Name</title>

</head>

<body>

<h1>Contact Me</h1>

<p>This is the contact page of your portfolio website.</p>

<p>Provide your contact information here.</p>

<a href="/">Home</a>

<a href="/about">About Me</a>

<a href="/portfolio">Portfolio</a>

</body>

</html>

Portfolio.html

<!DOCTYPE html>

<html>

<head>

<title>Portfolio - Your Name</title>

</head>

<body>

<h1>Portfolio</h1>

<p>This is the portfolio page of your portfolio website.</p>

<p>Showcase your projects and work here.</p>

<a href="/">Home</a>

<a href="/about">About Me</a>

<a href="/contact">Contact Me</a>

</body>

</html>

app.py:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('home.html')

@app.route('/about')

def about():

return render\_template('about.html')

@app.route('/portfolio')

def portfolio():

return render\_template('portfolio.html')

@app.route('/contact')

def contact():

return render\_template('contact.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

17. Design Blog app using flask.

Index.html:

<!DOCTYPE html>

<html>

<head>

<title>Blog</title>

</head>

<body>

<h1>Blog</h1>

<ul>

{% for post in posts %}

<li>

<h2>{{ post.title }}</h2>

<p>{{ post.content }}</p>

</li>

{% endfor %}

</ul>

<a href="/create">Create New Post</a>

</body>

</html>

Create.html:

<!DOCTYPE html>

<html>

<head>

<title>Create New Post</title>

</head>

<body>

<h1>Create New Post</h1>

<form action="/create" method="post">

<label for="title">Title:</label><br>

<input type="text" id="title" name="title" required><br>

<label for="content">Content:</label><br>

<textarea id="content" name="content" required></textarea><br>

<input type="submit" value="Create">

</form>

<a href="/">Back to Home</a>

</body>

</html>

app.py:

from flask import Flask, render\_template, request, redirect, url\_for

app = Flask(\_\_name\_\_)

# Temporary storage for blog posts (in-memory list)

posts = []

@app.route('/')

def index():

return render\_template('index.html', posts=posts)

@app.route('/create', methods=['GET', 'POST'])

def create():

if request.method == 'POST':

title = request.form['title']

content = request.form['content']

new\_post = {'title': title, 'content': content}

posts.append(new\_post)

return redirect(url\_for('index'))

return render\_template('create.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

18. Design BMI calculator Using flask

Index.html

<!DOCTYPE html>

<html>

<head>

<title>BMI Calculator</title>

</head>

<body>

<h2>BMI Calculator</h2>

<form action="/calculate" method="post">

<label for="weight">Weight (kg):</label><br>

<input type="number" id="weight" name="weight" required><br>

<label for="height">Height (m):</label><br>

<input type="number" id="height" name="height" step="0.01" required><br>

<input type="submit" value="Calculate BMI">

</form>

</body>

</html>

Results.html

<!DOCTYPE html>

<html>

<head>

<title>BMI Result</title>

</head>

<body>

<h2>BMI Result</h2>

<p>Your BMI is: {{ bmi }}</p>

<p>You are {{ category }}</p>

<a href="/">Back to Calculator</a>

</body>

</html>

app.py

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('index.html')

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

def calculate():

weight = float(request.form['weight'])

height = float(request.form['height'])

# Calculate BMI

bmi = weight / (height \*\* 2)

# Determine BMI category

if bmi < 18.5:

category = 'Underweight'

elif bmi < 24.9:

category = 'Normal Weight'

elif bmi < 29.9:

category = 'Overweight'

else:

category = 'Obese'

return render\_template('result.html', bmi=bmi, category=category)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)