Exercise1

void main() {

String zone = "ABC";

double weight = 3.0;

double costPerKg;

double totalCost;

if (zone == "XYZ") {

costPerKg = 5;

} else if (zone == "ABC") {

costPerKg = 7;

} else if (zone == "PQR") {

costPerKg = 10;

} else {

print("Invalid zone!");

return;

}

totalCost = costPerKg \* weight;

print("Zone: $zone");

print("Weight: $weight kg");

print("Shipping cost:${totalCost} dollars");

}

Exercise 2

void main(){

int totalFriends=8;

double billAmt = 6000;

double splitAmt=(billAmt)/(totalFriends);

print(" Split amount per person is ${splitAmt}");

}

Exercise 3

void main() {

List<int> a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89];

for (var n in a) {

if (n < 5) {

print(n);

}

}

}

Exercise 4

import 'dart:io';

import 'package:tuple/tuple.dart';

void main() {

stdout.write('Enter comma-separated numbers: ');

String? input = stdin.readLineSync();

if (input == null || input.isEmpty) {

print('No input provided.');

return;

}

List<String> numList = input.split(',').map((s) => s.trim()).toList();

// Supports up to Tuple7—so for more numbers you'd need a different approach

if (numList.length == 4) {

var numTuple = Tuple4(numList[0], numList[1], numList[2], numList[3]);

print('List: $numList');

print('Tuple: $numTuple');

} else {

print('Length must be 4 for Tuple4 example.');

}

}

Day2 Assignment

1.

import 'dart:math';

/// Generates a random password of given [length], mixing:

/// - lowercase letters (a–z)

/// - uppercase letters (A–Z)

/// - digits (0–9)

/// - special symbols (!@#$%^&\*()\_+ etc.)

///

/// Uses [Random.secure()] for cryptographic strength.

/// Throws [ArgumentError] if [length] is less than 1.

String generateRandomPassword(int length) {

if (length < 1) {

throw ArgumentError('Password length must be at least 1.');

}

const String lower = 'abcdefghijklmnopqrstuvwxyz';

const String upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

const String digits = '0123456789';

const String symbols = '!@#\$%^&\*()\_+-=[]{}|;:,.<>?';

final String allChars = lower + upper + digits + symbols;

final rand = Random.secure();

return List.generate(length,

(\_) => allChars[rand.nextInt(allChars.length)]).join();

}

2.

int sumEven(List<int> numbers) {

// Filter even numbers using an anonymous function (lambda), then sum them.

return numbers

.where((num) => num.isEven) // filters even values

.fold(0, (prev, curr) => prev + curr); // sums them up

}

void main() {

final list = [1, 2, 3, 4, 5, 6];

print(sumEven(list)); // Output: 12

}

3.

class Book {

String name;

String author;

double price;

Book(this.name, this.author, this.price);

void display() {

print('Book Name: $name');

print('Author: $author');

print('Price: \$${price.toStringAsFixed(2)}');

}

}

void main() {

final myBook = Book('1984', 'George Orwell', 9.99);

myBook.display();

}

4.

// Base class: Vehicle

class Vehicle {

String make;

String model;

Vehicle(this.make, this.model);

void displayInfo() {

print('Vehicle Make: $make, Model: $model');

}

}

// Subclass: Car extends Vehicle

class Car extends Vehicle {

int numberOfDoors;

Car(String make, String model, this.numberOfDoors)

: super(make, model);

void displayCarInfo() {

displayInfo(); // inherited method

print('Number of Doors: $numberOfDoors');

}

}

// Subclass: Motorcycle extends Vehicle

class Motorcycle extends Vehicle {

String engineType;

Motorcycle(String make, String model, this.engineType)

: super(make, model);

void displayMotorcycleInfo() {

displayInfo(); // inherited method

print('Engine Type: $engineType');

}

}

void main() {

// Create a Car instance

var myCar = Car('Toyota', 'Corolla', 4);

myCar.displayCarInfo();

print('---');

// Create a Motorcycle instance

var myBike = Motorcycle('Harley-Davidson', 'Sportster', 'V-Twin');

myBike.displayMotorcycleInfo();

}

5.

// Base class: Shape with a draw method

class Shape {

void draw() {

print('Drawing a shape.');

}

}

// Subclass: Circle overrides draw()

class Circle extends Shape {

@override

void draw() {

print('Drawing a circle');

}

}

// Subclass: Rectangle overrides draw()

class Rectangle extends Shape {

@override

void draw() {

print('Drawing a rectangle');

}

}

void main() {

// Create a list of Shape objects with both subclass instances

List<Shape> shapes = [

Circle(),

Rectangle(),

Circle(),

Rectangle(),

];

// Iterate and invoke polymorphic draw()

for (var shape in shapes) {

// Using runtimeType to include the object's actual type

print('Shape type: ${shape.runtimeType}');

shape.draw();

print('---');

}

}

6.

import 'dart:math';

/// A class-based approach to generate random passwords.

class PasswordGenerator {

final int length;

final bool includeSpecial;

// Define character sets

static const String \_lowercase = 'abcdefghijklmnopqrstuvwxyz';

static const String \_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

static const String \_numbers = '0123456789';

static const String \_special = '!@#\$%^&\*()-\_=+[]{}|;:,.<>?/~`';

String? \_password;

/// Constructor requiring desired [length] and whether to include [includeSpecial] characters

PasswordGenerator(this.length, this.includeSpecial) {

if (length < 1) {

throw ArgumentError('Password length must be at least 1.');

}

}

/// Generates a random password considering the configuration

String generatePassword() {

final String chars = StringBuffer()

..write(\_lowercase)

..write(\_uppercase)

..write(\_numbers)

..write(includeSpecial ? \_special : '')

.toString();

final rand = Random.secure();

\_password = List.generate(length, (\_) => chars[rand.nextInt(chars.length)])

.join();

return \_password!;

}

/// Prints the generated password, or prompts to generate one first

void displayPassword()

7.

Map<String, int> calculateSumAndProduct(List<int> numbers) {

if (numbers.isEmpty) {

return {'sum': 0, 'product': 0};

}

int sum = numbers.fold(0, (prev, curr) => prev + curr);

int product = numbers.fold(1, (prev, curr) => prev \* curr);

return {'sum': sum, 'product': product};

}

void main() {

var result1 = calculateSumAndProduct([2, 3, 5]);

print(result1); // Output: {sum: 10, product: 30}

var result2 = calculateSumAndProduct([1, 4, 7, 2]);

print(result2); // Output: {sum: 14, product: 56}

var result3 = calculateSumAndProduct([]); // Edge case: empty list

print(result3); // Output: {sum: 0, product: 0}

}

8.

// Represents a single to-do item

class TodoItem {

String title;

String description;

bool completed;

TodoItem({

required this.title,

required this.description,

this.completed = false,

});

void markCompleted() {

completed = true;

}

@override

String toString() {

final status = completed ? '[✓]' : '[ ]';

return '$status $title: $description';

}

}

// Manages the to-do list

class TodoList {

final List<TodoItem> \_items = [];

void addTodo(String title, String description) {

\_items.add(TodoItem(title: title, description: description));

}

void markTodoCompleted(int index) {

if (index < 0 || index >= \_items.length) {

print('Invalid index: $index');

return;

}

\_items[index].markCompleted();

}

void displayTodos() {

if (\_items.isEmpty) {

print('No to-do items found.');

return;

}

for (var i = 0; i < \_items.length; i++) {

print('${i + 1}. ${\_items[i]}');

}

}

}

void main() {

final myTodoList = TodoList();

// Add some tasks

myTodoList.addTodo('Buy groceries', 'Milk, Eggs, Bread');

myTodoList.addTodo('Walk dog', 'Evening walk in the park');

myTodoList.addTodo('Read book', 'Finish reading Chapter 4');

// Display initial list

print('--- Initial To-Do List ---');

myTodoList.displayTodos();

// Mark some as completed

myTodoList.markTodoCompleted(1); // completes 'Walk dog'

myTodoList.markTodoCompleted(0); // completes 'Buy groceries'

// Display updated list

print('\n--- Updated To-Do List ---');

myTodoList.displayTodos();

}