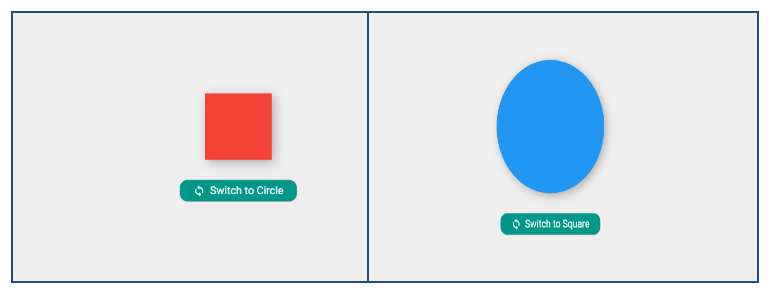
1. Display a container at the center of the screen.
2. Add a toggle button below the container to switch between two states:
   * State 1: The container is a small square (100x100 pixels) with a red color.
   * State 2: The container is a large circle (200x200 pixels) with a blue color.
3. Use the AnimatedContainer widget to smoothly transition between the two states over 800 milliseconds.
4. Ensure the animation includes changes to:

* Height and width.
* Border radius (to achieve the circle).
* Background color.

Main.dart:

|  |
| --- |
| import 'package:flutter/material.dart';  void main() {  runApp(AnimatedShapeApp());}  class AnimatedShapeApp extends StatelessWidget {  @override  Widget build(BuildContext context) {  return MaterialApp(  title: 'Animated Container Toggle',  debugShowCheckedModeBanner: false,  home: AnimatedShapeScreen(),);}}  class AnimatedShapeScreen extends StatefulWidget {  @override  \_AnimatedShapeScreenState createState() => \_AnimatedShapeScreenState();}  class \_AnimatedShapeScreenState extends State<AnimatedShapeScreen> {  bool \_isCircle = false;  void \_toggleShape() {  setState(() {  \_isCircle = !\_isCircle;});}  @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.grey[200],  body: Center(  child: Column(  mainAxisSize: MainAxisSize.min,  children: [  AnimatedContainer(  width: \_isCircle ? 200 : 100,  height: \_isCircle ? 200 : 100,  duration: Duration(milliseconds: 800),  decoration: BoxDecoration(  color: \_isCircle ? Colors.blue : Colors.red,  borderRadius: BorderRadius.circular(\_isCircle ? 100 : 0),  boxShadow: [  BoxShadow(  color: Colors.black26,  blurRadius: 12,  offset: Offset(4, 4),)],),  curve: Curves.easeInOut,),  SizedBox(height: 30),  ElevatedButton.icon(  onPressed: \_toggleShape,  icon: Icon(Icons.sync),  label: Text(\_isCircle ? "Switch to Square" : "Switch to Circle"),  style: ElevatedButton.styleFrom(  padding: EdgeInsets.symmetric(horizontal: 20, vertical: 12),  textStyle: TextStyle(fontSize: 16),  backgroundColor: Colors.teal,  foregroundColor: Colors.white,  shape: RoundedRectangleBorder(  borderRadius: BorderRadius.circular(12),),),)],),),); }} |

OUTPUT:



2: What is Unit Testing, and Why Is It Important in Flutter Development?

**Definition**

Unit Testing is the process of testing individual functions, methods, or classes in isolation to verify they work as expected.

**Importance in Flutter:**

* Catches Bugs Early: Ensures each function behaves correctly before UI is added.
* Fast Feedback: Unit tests are quick to run and don’t require UI rendering.
* Improves Code Quality: Encourages modular, testable code.
* Safer Refactoring: Lets you confidently make changes without breaking logic.
* Automation Ready: Ideal for CI/CD pipelines (automated builds & deployment).

2: How to Write a Unit Test for a Function That Calculates the Sum of Two Integers?

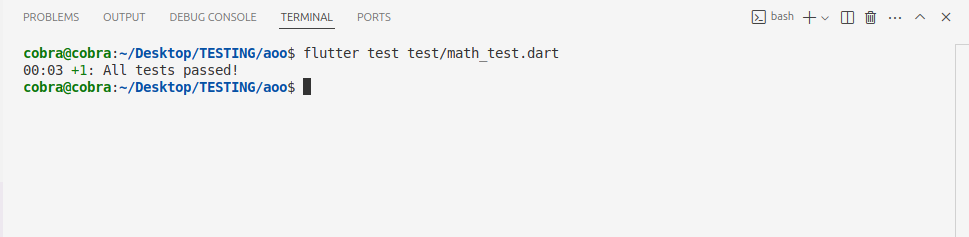
Ans:math.dart:

|  |
| --- |
| int add(int a, int b) => a + b; |

math\_test.dart:

|  |
| --- |
| import 'package:flutter\_test/flutter\_test.dart';  import 'package:aoo/math.dart';  void main() {  test('adds two numbers', () {  expect(add(2, 3), 5);  });  } |

Output:



3:Difference between Unit testing and Widget testing?

Ans:

Difference Between Unit Test and Widget Test in Flutter

**A)Unit Test:**

* Tests small pieces of logic (e.g., functions, classes).
* Does not depend on Flutter widgets or UI framework.
* Runs very fast.
* Easy to write and maintain.
* Example: Testing a sum(int a, int b) function.

**B) Widget Test:**

* ✔️ Tests Flutter widgets, UI layout, and interactions.
* ✔️ Depends on Flutter UI framework.
* ✔️ Slower than unit tests (needs to render widgets).
* ✔️ Useful for checking how UI responds to user actions.
* ✔️ Example: Testing if tapping a button increases a counter.