COLLEGE OF SCIENCE AND TECHNOLOGY
SCHOOL OF ICT
COMPUTER ENGINEERING
LEVEL 3

GROUP 1:
MEMBERS: 1. REOBOTH IRADUKUNDA 22302668
2. VALENTIN MUHIRE 22302677
3. DAVID RUBIMBURA 22101456

REPORT FOR LAB 1

(Q1) A Dart function named WelcomeMessage that prints a welcome message for the school system.

what the Code does:

The Code Creates Dart function Called welcomeMessage that displays a formatted welcome message for a school system. The function prints a bordered greeting with the school's welcome message and mission statement, Creating a professional and Consistent introduction for all users of the system.

Key Concepts used:

1. Function definition: Using void welcomeMessage() to declare a function that performs an action without returning a value.
2. print() Statements: multiple print statements to display formatted text output with borders and spacing.
3. Code organisation: Separating the welcome message logic into its own reusable function.
4. Main function: Demostrating how to Call the function from the program's entry point

what was learned;

1. Class structure: we learned how to define classes with properties and Constructor
2. Data Encapsulation: We understood how

1. Basic function syntax: we learned how to properly define a function in Dart with the Correct return type and structure.

3. Named parameters make function calls more descriptive and self-explanatory
4. Required vs optional: using required keyword ensures critical parameters aren't accidentally omitted.

Why Named parameters are Helpful:

Named parameters are helpful because they make function calls more readable and self-documenting. when calling CreateStudent(name: 'Alice', age: 16), it's immediately clear what each value represents without needing to check the function definition. This is especially valuable when functions have multiple parameters, when parameters have similar data types or when you want to skip optional parameters. they also make code maintenance easier because you can add new parameters without breaking existing function calls.

Q3) A function named CreateTeacher with one required parameter name and one optional parameter Subject

What Code does:

The code Creates a CreateTeacher function with one required parameter name and one optional parameter Subject that defaults to subject not assigned

Key Concepts:

1. Required parameter: name must be provided
2. optional parameter: Subject in [] can be omitted
3. Default handling: shows "Subject not assigned" if no Subject given

What we learned:

1. optional parameters use square brackets [ ]
2. Functions can have both required and optional parameters
3. Use ?? or if-else to handle missing optional values
4. optional parameters must come after required parameters.

(Q5) An object of student and print the student's name and age.

What the code does:

Creates an object with name and age. prints the name and age to verify the object was created Correctly

Key Concepts:

1. object Creation: Student ('Emily wilson', 16) Calls the Constructor
2. Variable storage: Student student = ... stores the object
3. Dot Notation: student.name and student.age access properties
4. output: print() displays the values

What we learned:

1. How to instantiate an object from a class
2. Objects hold specific data for each instance
3. Dot notation reads property values
4. Each new object has its own copy of data

Object Creation and Usage:

Object Creation turns the class blueprint into a real instance with actual data. The Constructor runs and sets property values. Once created, the objects exists in memory and can be accessed using its variable name, properties are read using dot notation, this allow us to create many different students from the same Student class, each with their own name and age.

(Q6) Create a class person with a variable name and a function introduce() that prints the name

what the code does:

Creates a person class with name property and introduce() method that prints "Hello, my name is ---", shows how classes can have both data and behaviour.

What we learned

1. extends makes one class inherit from another
2. super passes values to the parent Constructor
3. Child class has everything parent has, plus its own
4. Common code stays in parent, specific code in child

How inheritance works:

student "is-a" person, student automatically gets all person's properties and methods without rewriting them. student adds its own features while reusing person's code. this prevents duplication and creates logical relationship between classes.

(Q8) Create an abstract class 'Registrable' with a function 'registerCourse (String courseName)'.

what the code does:

Creates an abstract class Registrable with one method registerCourse (String courseName). No implementation. Just a Contract. Cannot Create objects from it

key concepts:

1. Abstract class: abstract class cannot be ~~implemented~~ instantiated
2. Abstract method: method with no body {}
3. Contract: forces implementing classes to provide this method
4. Interface: defines what to do, not how to do it

what we learned:

1. abstract keyword creates classes that can't be instantiated
2. Abstract methods end with; not {}
3. child classes must override and implement abstract methods
4. Separates method declaration from implementation

Q10. Create a mixin AttendanceMixin that stores an attendance counter and has a function markAttendance() to increase attendance

What the code does:

Creates a mixin AttendanceMixin with attendance counter and markAttendance() method. Package attendance tracking as reusable behavior.

Key concepts:

1. Mixin: mixin AttendanceMixin defines reusable code.
2. Private Variable: _attendanceCount stores data, can't be accessed directly.
3. Method: markAttendance() increments counter.
4. Getter: attendanceCounter reads private counter value.

What we learned:

1. mixin keyword creates reusable behavior package.
2. _makes member private to the library.
3. Getter provide controlled access to private data.
4. Mixin are not classes - can't be instantiated alone.

What a mixin is:

A mixin is a bundle reusable code that can be added to any class with. Both not inheritance. Add capabilities, not parent relationship. 'Has-a' not 'is-a' Multiple mixin can be added to one class. No complex hierarchy. Like adding skills to a character. Different classes can share same mixin without being related.

Q11.

Apply AttendanceMixin to student. Mark attendance 3 times and print the attendance

What the code does:

Adds AttendanceMixin to student class using with. Creates a student object, marks attendance 3 times, prints total attendance.

Key concepts:

1. Mixin Application: with AttendanceMixin adds mixin to class.
2. Method Access: student markAttendance() calls mixin method directly
3. Counter Tracking: Attendance increments each time.
4. Output: print('student.attendanceCount') shows total

Q13. Create a map where the key is student ID and value is a Student. Print all student names.

What the code does:

Creates a Map where student ID (String) is the key and Student object is the value. Prints all student Names by iterating through the map.

Key Concepts:

1. Map<KeyType
   1. Map<String, Student> stores key-value pairs
   2. Key: Student ID used to look up students
   3. Value: Student object stored with its ID
   4. Iteration: forEach() loops through all entries

What we learned

1. Map<KeyType, ValueType> defines map type
2. Keys must be unique, values can repeat.
3. forEach((key, value) {}) iterates through map.
4. Maps use keys, Lists use indexes

Maps and when they are useful

Maps store data as key-value pairs. Look up values by key, not position. Key is like an ID, value is the data. Useful when you need to find something by its identifier. Student ID → Student, employee ID → Employee, country code → Country. Faster than searching through a list. Direct access by meaningful key.

Q14. Use an anonymous function to print all student names from the list

What the code does:

Uses forEach() with an anonymous function to loop through the student list and print each student's name. Function has no name, defined inline.

Key Concepts:

1. Anonymous function: Function without name (Student) {...}
2. forEach: List method that runs a function on each element.
3. Inline declaration: Function defined directly where it is used.
4. Callback: Passing behavior as an argument.

Question 15: Write an arrow function that takes a student name and Prints a greeting message

What Code does:

Creates greetStudent() using arrow Syntax => Takes name Parameter, Prints welcome message. Single expression, no braces or return

Key Concepts:

1. Arrow function: => replaces {} and return
2. Single expression: one line of code after arrow
3. Parameter: (string name) function input
4. Print Statement: Directly Prints greeting

What We learned

1. => makes functions shorter and cleaner
2. No curly braces {} needed
3. No return keyword - expression is automatically returned
4. Best for simple, one-line functions

Arrow functions and Simplicity:

Arrow Functions are shorthand for functions with one expression. No {}, no return. The expression after => is automatically returned. Makes simple function easier to read and write. Use for small operations, not complex logic. ~~Clear funct~~
cleaner than traditional function Syntax

Question 16: Write an async function loadStudent() that waits 2 seconds and return the list of students.

Key Concepts:

1. await: await Pauses until future Completes
2. Async Main: main() async allows await inside main
3. Result Capture: List<Student> Students = await loadStudents()
4. Output: Prints Students.length to Show Count

What he learned:

1. main() Can be async with main() async
2. await extracts values from Future
3. Code after await runs only after future Completes
4. Makes async Code look like normal Step-by-Step Code

How Async Helps in Real Apps:

Real apps wait for internet, files, databases. Without async, app freezes while waiting, with async, app Stays responsive. User Can Still tap buttons, Scroll, Cancel. No spining beachballs. Essential for Flutter, web, mobile, Async/await keeps Code Simple While Keeping apps fast and responsive

Question 18: Explain in your own words: Why mixins are useful, and how inheritance and mixins are different

Why mixins are useful

1. Code Reuse aCross Hierachies: Share functionality between unrelated classes
2. Avoid domain Problem: No ambiguity when Same method exist in multiple Parents
3. Flexible Composition: Add behaviours like building blocks, not rigid hierachies
4. Single Responsibility: Keep classes focused, mixins handle Cross-Cutting Concerns
5. Testability: mixins Can be tested Independently

3. Method Call : SendCourseNotification() Called inside registerCourse()
4. Seamless Integration : Mixin method works like class method

What We learned

1. Multiple mixins added with Commas : with Mixin1, Mixin2
2. Mixins work independently of each other
3. Features can be added without touching Inheritance
4. Mixins can be mixed and matched Per class

How the New Mixins was used :

Added NotificationMixin to Student with. Now Student has both attendance tracking and notification Sending. No inheritance needed. No code duplication Just add the Capability. Clean, modular, reusable. RegisterCourse → Notification Prints automatically

Question 20 : Write a short paragraph Explaining how learning Dart helps You understand Flutter

Dart is the foundation of flutter development. By learning Dart, we gain:

1. WIDGET UNDERSTANDING: - Flutter widgets are Dart classes
   - State management uses Dart's reactive Programming
2. ASYNC OPERATIONS : - Flutter appas heavily use async/await for API Calls
   - Future and Stream are core Dart Concepts
3. OBJECT-ORIENTED STRUCTURE : - Flutter's widget tree is built using Dart OOP
   - Inheritance and mixins Create reusable UI Components
4. TYPE SAFETY : - Dart's Strong typing prevents runtime errors in Flutter
   - Null Safety ensures robust mobile applications