UNIVERSITY OF RWANDA
COLLEGE OF SCIENCE AND TECHNOLOGY
SCHOOL OF CCI
DEPARTMENT OF COMPUTER AND SOFTWARE ENGINEERING
ACADEMIC YEAR: 2025-2026
YEAR OF STUDY: III
MODULE: MOBILE APPLICATION SYSTEMS AND DESIGN
TASK: DART PROGRAMMING LAB 1

Date: On Friday, Feb. 06th, 2026

GROUP MEMBERS:
1. Fabrice NDAYISABA 223008047
2. Silas HAKUZWIMANA 223001019

LAB QUESTION SUMMARIES

PART 1: FUNCTIONS

Q1. A function is a reusable block of code that performs a specific task.
We created this to provide a standard greeting that can be called any where in the app/project without rewriting the print statement.

Q2. Named parameters are enclosed in curly brackets ({}), they are helpful because they make the code readable by labelling arguments (e.g. name: "Clever") and allow parameters to be passed in any order.

Q3. Optional parameters are enclosed in square brackets ([]). they allow a function to be called with or without that specific argument. If omitted, the variable becomes null or uses a default value.

PART 2 & 3: CONSTRUCTORS, CLASSES, AND INHERITANCE

Q4. A constructor is a special method in a class that is automatically called when an object is created. It is used to initialize the properties of the class.
Constructors ensure objects start with correct data.

Q5. An object is created from a class. It is an instance of a class.
The "student" object stores and displays data.
objects allow us to use class properties.

Q6. A class is a blueprint for creating objects.
It defines variables and methods.
classes help organize code.

Q7. Inheritance allows a class to reuse another class.
"student" class inherits from "person" class.
It reduces code duplication. through allowance to use variable and methods of parent class to child class.

PART 4: INTERFACES

Q8. An interface defines methods a class must implement
It ensures consistency.
interfaces act as rules.
It is also defined as contract. defined via abstract classes in part.

Q9. Implementing an interface forces a class to define specific methods, ensuring that all "Registerable" objects follow the same rules.
student implements Registerable.
This enforces structure.

PART 5: MIXINS

Q10. A mixin adds extra behaviour to a class.
It allows code reuse without inheritance
Mixins are flexible.
In our lab, it plugs in extra behaviour like counting attendance to a class.

Q11. The mixin was applied to "student".
Attendance was increased and printed.
Mixins extend class functionality.

PART 6: COLLECTIONS

Q12. Lists are ordered groups of items used for sequencing.
It was used to stored Students.
It manages collections of data.

Q13. Maps are collections of key-value pairs, useful for looking up data quickly using unique identifier (like student ID).

PART 7: ANONYMOUS AND ARROW FUNCTIONS

Q14. Anonymous functions are functions without a name, used for short-lived tasks and reduces extra code.

Q15. Arrow functions are a short-hand syntax for functions that contain only one expression.
They make code simple and readable.

PART 8: ASYNCHRONOUS PROGRAMMING

Q16. Async/Await allows a program to handle tasks that take time (like loading data) without stopping the entire app.
* Async functions allow delayed operations.
* They prevent blocking the program.
* They improve performance.

Q17. "Await" waits for an async task to finish.
It ensures correct execution order.
Async improves real applications, keeping apps responsive while waiting for a database or server response.

PART 9: INTEGRATION CHALLENGE

Q18. Mixins are useful for adding specific "skills" to a class without forcing it into a family tree. Inheritance represents an "is-a" relationship (a Student is a Person), while a mixin represents a "can-do" relationship (a Student can mark attendance)

Key take aways:
* Mixins add behaviour, inheritance defines relationship.
* Mixins are flexible.
* they serve different purposes.

Q19. We created NotificationMixin to handle the logic of alerting the system. By applying it to "Student", the student object gains ability to trigger a notification automatically upon registration.

⇒ Mixins add features easily.

Q20. Dart is the language used by Flutter.

By learning Dart's OOP (classes, inheritance) and Asynchronous patterns I can understand how Flutter widgets work and how to handle user data/API calls effectively.

Dart's structure is the foundation for everything seen on Flutter screen.

↳ Understanding Dart makes Flutter easier.

↳ It improves mobile app development.