UNIVERSITY OF RWANDA

COLLEGE OF SCIENCE AND TECHNOLOGY

GAKO CAMPUS

COMPUTER ENGINEERING

MOBILE APPLICATIONS

DATE ON 15ty feb 2026

Group members : -Abdoul SIBO 223026681

-JUSTINE IRAFASHA 223026693

-FILS NIYINZIMA 223026876

DART LAB REPORT (lab 1)

scanned document explanations.

UR-CST
GAKO - Campus
CE - Lev III

On 15th Feb 2026

- Module : Mobile Communication

Grp Members:
- Abdol Siko - 223026681
- Justine Irafasha 223026693
- Filg Niyonzima 223026876

Assignment 1

part A : Functions

Function welcome Message

(1) Void welcomeMessage() {
  print ("Welcome to the School Management system!");
}

Summary / Report

A function is a reusable block of code that performs a specific task.
The welcomeMessage function was created to display a greeting message for the school system.

Using a function avoids repeating the same code and makes the program easier to maintain and understand.

(2) Function create student (Named parameters)

Void createStudent ({required string name, required int age}) {
  print ("Student Name : $name");
  print ("Student Age : $age");
}

Summary / Report

Named parameters allow values to be passed to a function by specifying their names.
They improve code readability and reduce errors because it is clear what each value represents.

In the create student function, named parameters are used to clearly identify the student's name and age.

③ Function createTeacher (optional parameter)

```
void createTeacher (String name, [String ? subject]) {
print ("Teacher Name : $name");
if (subject != null) {
  print ("subject : $ Subject");
} else {
  print ("subject not assigned");
}}
```

Summary / Report

Optional parameters allow a function to be called with or without certain values.

In Dart, optional parameters are placed inside square brackets [].

In the createTeacher function, the subject parameter is optional.

If it is not provided, the program prints "Subject not assigned", making the function flexible and user-friendly.

Part 2 : constructors and classes

④ Student class with constructor

```
class Student {
 String name;
 int age;
 Student (this.name, this.age);
}
```

Summary / Report

A constructor is a special function that runs automatically when an object is created.

It is used to initialize variables with initial values.

The Student constructor ensures that every student object has a name and age when it is created, making the object complete and valid.

⑤ create a Student object and print Details

```
Void main () {
  Student student1 = Student ("Alice", 20);
  print (Student1.name);
  print (student1.age);
}
```

Summary | Report

An object is an instance of a class.

Object creation allows us to use the properties and methods defined in a class.

Here, Student1 is created from the student class and used to access and print student details.

Part 3 : Inheritance

⑥ Person class

```
class Person {
  String name;
  Person (this.name);
  void introduce () {
  print ("My name is $name");
  }}
```

Summary | Report

A class is a blueprint used to create objects.

It defines properties (variables) and behaviors (functions).

The Person class represents a general human with a name and an introduction method.

⑦ Student Inherits from Person

```
class Student extends Person {
  int age;
  Student (String name, this.age): super(name);
}
  void main () {
Student student = Student ("Alice", 20);
student.introduce ();
}
```

Summary | Report

Inheritance allows a class to reuse properties and methods of another class.

The Student class inherits from Person, so it can use the introduce() function without rewriting it.

This reduces code duplication and improves reusability.

Part 4: Interfaces

⑧ Abstract class (Interface)

```
abstract class Registrable {
  void registerCourse(String courseName);
}
```

Summary | Report

An interface defines a set of rules that a class must follow.

In Dart, abstract classes are used as interfaces.

They ensure that any class implementing them provides required methods.

⑨ Student Implements Registrable

```
Class Student extends Person implements Registrable {
  int age;
Student(String name, this.age) : super(name);
  @override
void registerCourse(String courseName){
  print("$name registered for $courseName");
}}
```

Summary | Report

Implementing an interface forces a class to follow specific rules.

The Student class must implement registerCourse(), ensuring consistency and reliability in large applications.

Part 5 : Mixins

(10) Attendance Mixin

```
mixin AttendanceMixin {
int attendance = 0;
void markAttendance () {
    attendance ++;
}}
```

Summary | Report

A mixin is a reusable class that adds functionality to other classes. It allows sharing behavior without using inheritance.

(11) Apply AttendanceMixin to Student

```
class Student extends Person with AttendanceMixin Implements
                                    Registrable {
    int age;
    Student ( String name, this. age): super(name);
}
Void main ( ) {
Student student = Student ("Carine", 20);
student. markAttendance ( );
student. markAttendance ( );
student. markAttendance ( );
print ("Attendance : ${ student . attendance}");
}
```

Summary | Report

Mixins add extra behavior to a class without changing its main structure.

AttendanceMixin allows the Student class to track attendance without inheritance.

Part 6 : Collections

(12) List of Students

```
List < Student > students = [
    Student ("Alice", 19),
    Student ("Bob", 21);
    Student ("Chris", 20);
];
```

Summary / Report

A List stores multiple values in an ordered way. Lists are useful when handling collections of similar objects like students.

(13) Map of Students

```
Map<int, Student> studentMap = {
  1: Student("Alice", 19),
  2: Student("Bob", 21),
  3: Student("Chris", 20),
};
studentMap.forEach((id, student){
  print(student.name);
});
```

Summary / Report

A Map stores data in key-value pairs.
Maps are useful when data needs to be accessed quickly using unique identifiers like student IDs.

Part 7: Anonymous and Arrow Functions

(14) Anonymous Function

```
students.forEach((student){
print(student.name);
});
```

Summary / Report

An anonymous function has no name and is used for short tasks. It is commonly used with collections for quick operations.

(15) Arrow Function

```
void greetStudent(String name) => print("Hello, $name!");
```

Summary / Report

Arrow functions provide a shorter and cleaner way to write simple functions.
They improve readability and reduce code length.

Part 8: Asynchronous programming

(16) Async Function loadStudents()

```
Future<List<Student>> loadStudents() async {
await Future.delayed(Duration(seconds: 2));
return students;
}
```

Summary | Report

Async functions allow programs to wait for long tasks without stopping execution.

The await keyword pauses execution until the task is completed.

(17) Call Async Function in main()

```
void main() async {
List<Student> loadedStudents = await loadStudents();
print("Students loaded: ${loadedStudents.length}");
}
```

Summary | Report

Asynchronous programming helps real apps stay responsive.

It is useful for loading data from databases, APIs, or files without freezing the application.

Part 9: Integration Challenge

(18) Mixins are useful because they let programmer share behavior across unrelated classes without forcing a rigid parent-child tree.

They are a form of composition that keeps your code modular, reusable, and expressive. Inheritance defines what something is; mixins define what it can do.

(19) NotificationMixin

```
mixin NotificationMixin {
  void notify(String message){
  print(message);
}}

class Student extends Person
with AttendanceMixin, NotificationMixin
implements Registrable {

int age;

Student(String name, this.age): super(name);
@override
void registerCourse(String courseName){
  notify("$name has successfully registered for $courseName");
}}
```

Summary / Report

The new mixin adds notification behavior to the Student class. It allows sending messages without modify inheritance structure.

(20) Dart is the language Flutter is written in. To use Flutter, you write Dart code. If you know Dart, you already understand the basics Flutter needs - like classes and functions. you stop struggling with the language and can focus on building the app.