UR - CST

Gako Campus

Computer Engineering

Year 3

Mdoile Application

Group Members: Janvier NSHIMIYIMANA 22302667 9

Blandon ASHIMIRWE 22 30 26675

Gilbert HIRWA NIWALI 22302668 9

date: 13/02/2026

LAB 1

PART 1: FUNCTIONS

Q1. Welcome Message Functions

```
void welcomeMessage()
{
print (" welcome to the school Management system! ");
}
```

- A function is a block of code that performs a speafic task. The function was created to display a welcome message whenever a program starts. it helps to avoid repeating the same code.

Q2. Function with named parameters

```
Void create student ( { required string name , required int age })
{ print (" student Name : $ name ");
  print (" student Age : $ age ");
```

Named parameters allow us to pass value by using paramater names. They make the code easier to read and prevent confusion about the order of arguments.

the "required" keyword ensures these parameters must be passed when calling the function.

3. Function with required and optional positional parameter.

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

Here the optional parameter are enclosed in square brackets [ ] which is "subject" if not provided, it default to null. This proves flexible when not all information is available at the time of creation.

PART 2: CONSTRUCTORS AND CLASSES

Q4. Student class with constructor

```
class student {

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

A constructor is a special method that is automatically called when an object is created. It is used to initialize the object's properties.
Constructors are important because they ensure an object starts in a valid with all required data.

Q5. Create student object

```
void main()
Student student1 = student ("Ali", 20);
Print ("Name : ${student1.name}");
print ("Age : ${student1.age}");
```

Object is an instance of class.
We create objects to use the class features and access its available and methods. the object student1 will access all features of class student.

PART 3 INHERITANCES

Q6 Base class Person.

```
class person {
    string name = '';

  Void introduce ()
          {
      print ('Hello, my name is $ name.');
          }
    }
```

Class is a blueprint used to create objects.
It defines variable and functions that describe an object.

Q7. student inherits from Person.

```
Class student extends Person {
      int age;
  student( this.age, string name)
          {
    this.name = name;
        } }
```

Inheritance one class to reuse another Class's properties and methods.
student inherits name and introduce() from person so it doesnot need to rewrite them.

PART 4 INTERFACES

Q8. Abstract class (interfaces)

```
Abstract class Registrable {
      void registerCourse (string CourseName);
          }
```

An interface is a set of rules that class must follow
In Dart, an abstract class can act as interface.
It tells other classes what methods they must implement

Q9. Implementing the Registrable in student.

Code :

```
Class student implements Registrable {
        string name;
        int age;
    Student (this.name, this.age);

@override
    Void registerCourse (string courseName) {
    print('$name has registered for $.courseName');
                }
}
```

when a class implements an interface, it promises to provide implementation for all methods in that interface.
the "implement" keywords is used and @override indicates we are providing our own version of the method.
This enforces a standard: any class that is Registrable must have a registerCourse() method..

PARTS: MIXINS

Q10. Mixins allow you to reuse code in multiple classes hierarchies without using inheritance.

Code to create Mixin

```
mixin AttendanceMixin {
    int attendanceCount = 0;
    Void markAttendance() {
        attendanceCount ++;
    } }
```

Q11. Apply AttendanceMixin to student:

Code :

```
Class studentwithAttendance extends with AttendanceMixin {
    studentwithAttendance (string name, int age) : super (name, age); }
```

The mixins add behaviour horizontally, unlike inheritance which adds vertically. This is useful when multiple unrelated classes need share function

PART 6: COLLECTIONS

Q12. List storing multiple 'student' objects. Add 3 students

```
Void ListExample () {
    List < Student > students = [
                student ('Alice', 20),
                student ('Bob', 22),
                student ('Charlie', 19) ];

    // For adding students
    students.add (student ('Kevin', 24));
    students.add (student ('Eric', 25));
    students.add (student ('Filk', 20));

    print (students);
```

List is an ~~array~~ ordered group of items. They are useful when you need to keep data in sequence. It is index based collection of objects.

Q13. Map where the key is student ID and value is student.

Code:

```
print ('student Map ...');
Map < string, student > studentMap = {
    'S001' : student ('Alice', 15),
    'S002' : student ('Bob', 16),
    'S003' : student ('Charlie, 17), };

Print ('the student names are:');
for (var student in studentMap.value) {
        print (student.name); } }
```

A Map stored data in key (value pairs). Maps are useful when each item has unique ID, fast searching is needed and data must be accessed by key. the above code, students IDs are used as key.

PART 7: ANONYMOUS AND ARROW FUNCTIONS

Q14. Anonymous function to print all student names from the list.

```
Void PrintStudentNames ( List < student > students )
                        {
students. For Each ( (student) {
                print(student. name); } );
                        }
```

An anonymous function is a function without name and are define inline.
they are used for quick task that are passed as arguments to other functions.

Q15. Arrow function that takes students name and prints greeting message.

```
Var greeting = ( string name ) => print (" Hello, welcome , $name !");
        greeting ('Kalisa');
```

Arrow functions (=>) are a shorthand for functions that contain only one expression. they make code cleaner and more readable.

PART 8: ASYNCHRONOUS PROGRAMMING

Q16. An async function 'load student()' that waits 2 second, and returns list of students

```
Future < List < student >> loadstudents () async {
 await Future. delayed ( Duration (seconds :2)) ;
 return [ student ('Alice', 20), student ('Bob', 22), student ('Charlie', 19)]
                        }
```

Async functions allow programs to wait without stopping.
It use async keyword
Use wait to pause
help handle slow tasks.

Q17. ~~Void~~ Main call.

Code: void main() async {

List<student> loadedList = await loadStudents();

print('successfully loaded ${loadedList.length} students.');

}

In real apps, async/wait prevent the UI form freezing. It allows the app to say responsive while waiting for data from the internet.

PART 9: INTEGRATION CHALLENGE

Q18. Mixins are useful because they allow us to reuse code on multiple classes without creating parent-child relationship.

Inheritance create a strong relationship where one class depends on another but mixin only add features.

Q19. Mixin NotificationMixin {

Void SendNotification (string course) {

print('Notification: you are registered in $course');

}

}

Class student with NotificationMixin {

String name;

int age;

student(this.name, this.age);

Void registerCourse(string course) {

print("$name registered for $course");

sendNotification(course);

}}

⇒ The NotificationMixin adds a notification feature to student.

It used by
adding with NotificationMixin
calling sendNotification()

This allow the student to receive messages without rewriting code.

Q20. Dart is ~~the~~ engine behind flutter. Understanding Dart features like classes, Mixins and async programming is essential because everything in flutter is a class (widgets). If you know how Dart handles data and functions, building the UI in flutter become much easier because you understand how data flows through the app.