Part 1: functions

Q1: Dart function named "welcomeMessage" that prints a welcome message for the School system

```
Void main() {
    welcomeMessage();
}
Void welcomeMessage() {
    print("welcome to the School Management System");
}
```

Comment: we created this function to become reusable with block of codes that are performing task of displaying a welcome message of welcome whenever the system starts.

Q2: function named "createStudent" that uses named parameters ('name' and 'age')

```
Void main() { createStudent(name: 'John Doe', age: 20); }
Void createStudent({required String name, required int age}) {
    print("Student Name: $name, Age: $age");
}
```

we learnt that name parameters allow values to be passed using their parameters names

this make function to call easier and reduce errors.

Q3: Write a function named "CreateTeacher" with one required parameter 'name' and one optional parameter 'subject'

```
Void CreateTeacher(String name, [String? subject]) {
print("Teacher Name: $name");
print("subject: ${subject ?? 'subject not assigned'}");
}
```

* from optional parameters, we learnt that a function can be allowed to work even if some values are not provided, where by if the subject is not given, a default message is printed making the function flexible.

Part 2: Constructors and Classes

Q1: Class named 'student' with 'name' and 'age'

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

* We learnt that Constructor ensures that objects start with valid and meaningful data. This is to mean that a Constructor used to initialize class variables when an object is created.

Q5: and object of 'Student' and print sudent's name and age.

```
Void main (): {
    Student studentA = Student ("Alice", 20);
    print ("Name: ${studentA.name}, Age: ${studentA.age}
    );
}
```

* From this Question we created an object from class (StudentA from Student). which acted as an instant of class and allowed us to use the properties and methods defined in Class Student.

Part 3; Inheritance

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

```
Class person {
    String name;
    person (this.name);
    Void introduce() {
        print ("my name is $name");
    }
}
```

from this task we understood that Class define properties variables and behaviors of how our program will perform which means it is like a blueprint used to create objects

Q7: Make 'student' inherit from 'person' and call 'introduce()' from a 'student' object.

```
Class person {
    String name;
    person(this.name);
    Void introduce() {
        print("Hello, myname is $name");
    }
}

Class student extends person {
    Int age;
    Student(String name, this.age) : super(name);
}

Void main() {
    Student s = Student('billy', 23);
    s.introduce();
}
```

Comments: Students Reuse person's field and methods, so Students instances can call introduce() without reimplementing it.

part 4 : Interfaces

Q8: abstract class 'Registrable' write function 'registerCourse'

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

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

Q9: Make 'Student' implement 'Registrable' and Implement 'registerCourse' to print the student name and Course.

```
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");
    }
}
```

an interface forces a class to define required methods.

Part 5: Mixins

Q10: Attendance Mixin that stores an attendance counter

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

* A mixin is a way to reuse code across multiple classes without inheritance.

Q11: Apply MixIN to 'student'

```
Class Student extends person with AttendanceMixin {
    int age;
Student (String name, this.age) : super (name);
}

Void main () {
    Student student = Student ("billy", 22);
    Studet.markAttendance ();
    student.markAttendance ();
    Studet.markAttendance ();
    print ("attendance: ${student.attendance}");
}
```

Mixins adds extra behavior to a class without changing its inheritance structure.

Part 6 : Collections

Q12: Create a List storing multiple 'Student' objects.

```
List<Student> students = [
    Student("Alice", 20),
    Student("Bob", 21),
    Student("Charlie", 22),
];
```

* Lists store multiple values in an ordered ways. to manage collections of objects.

Q13: Map.

```
Map<int, Student> studentMap = {
    1: Student("Alice", 20),
    2: Student("Bob", 21),
    3: Student("Charlie", 22),
};
studentMap.forEach((id, student) {
    print(student.name);
});
```

* Maps stores data as key-value pairs

Part 7: Anonymous functions

Q14: Anonymous function

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

* anonymous function improve code readability.

Q15: arrow function that takes a student name and prints a greeting message

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

* arrow function provide short syntax for simple functions, which makes code clean and readable.

Part 8: Asynchronous programing

Q16: async function `loadStudents()` that waits 2 seconds and returns the list of students.

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

* async function allow the program to wait for long tasks without freezing the app.

Q17: In main(), call `loadStudents()`, use await, and print the number of students loaded.

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

* Async programing helps applications remains responsive while performing background tasks like loading data.

Part 9: Integration Challenge

Q18: MIXINS are useful because they allow a class to reuse certain behaviors without forcing it to inherit from another class. with mixins, I can add extra features Like attendance tracking or notifications to a class without changing its main Structure. This helps keep the code flexible and avoids repeating the same code in many classes.

The difference between Inheritance and mixins is that Inheritance creates a parent-child relationship between classes, where a child class automatically gets all properties and methods of the parent class.
Mixins, on the other hand, are used to add specific behaviors to a class without forming a parent-child relationship. Inheritance is used when classes are closely related, while mixins are used when we just want to share functionality.

Q19: new mixin 'NotificationMixin'

```
mixin notify(String message) {
  print("notification: $message");
    }
  }
```

```
Class Student extends person
  with attendanceMixin, NotificationMixin
  Implements Registrable{

    int age;

Student (String name, this.age): super(name);

@override

Void register Course(String courseName) {
  print("$name registered for $courseName");

notify("$name sucessfully registered for $courseName");

    }
  }
```

Q20: Learning Dart helped us understand flutter because Dart is the main language used to build flutter applications when we learn Dart, I understand how variables, functions, Classes, and Widgets work in flutter.

Date: 06/02/2026

Department: CSE

Module TITLE: Mobile Applications

REG No: - 222 013795 Zinanyiza Billy Charmant

- 223003227 Phillipe Mupisha.