Q:1 Create an empty list of type String. Name it months. Use the add method to add the names of the twelve months.

```dart
void main() {
  List<String> month = [];
  month.add("jan");
  month.add("feb");
  month.add("March");
  month.add("April");
  month.add("May");
  month.add("Jun");
  month.add("July");
  month.add("aug");
  month.add("sep");
  month.add("oct");
  month.add("nov");
  month.add("dec");
  print(month);
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 1.dart
[jan, feb, March, April, May, Jun, July, aug, sep, oct, nov, dec]

Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$
```

Q: 2. Make an immutable list with the same elements as in Mini-exercise 1

```dart
void main() {
  final List<String> month = [
    "jan",
    "feb",
    "March",
    "April",
    "May",
    "Jun",
    "July",
    "aug",
    "sep",
    "oct",
```

```
    "nov",
    "dec"
  ];
  print(month);
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 2.dart
[jan, feb, March, April, May, Jun, July, aug, sep, oct, nov, dec]
```

Q 3. Use collection for to create a new list with the month names in all uppercase.

```
void main() {
  final List<String> month = [
    "jan",
    "feb",
    "March",
    "April",
    "May",
    "Jun",
    "July",
    "aug",
    "sep",
    "oct",
    "nov",
    "dec"
  ];
  List<String> monthsInUppercase = [
for (var monthh in month) monthh.toUpperCase()
];
print(monthsInUppercase);
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 3.dart
[JAN, FEB, MARCH, APRIL, MAY, JUN, JULY, AUG, SEP, OCT, NOV, DEC]
```

Exercise : 02 :

1. Create a map with the following keys: name, profession, country and city. For The values, add your own information.

```
2. void main(){
3.
4.     Map<String, String> personalInfo = {
5. 'name': 'rajan patel',
6. 'profession': 'Computer Engineering Student',
7. 'city': 'navsari',
8. 'country': 'India'
9. };
10.  print(personalInfo);
11.
12.  }
13.
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 4.dart
{name: rajan patel, profession: Computer Engineering Student, city: navsari, country: India}
```

2. You suddenly decide to move to Toronto, Canada. Programmatically update the values for country and city.

```
void main(){

    Map<String, String> personalInfo = {
            'name': 'rajan patel',
            'profession': 'Computer Engineering Student',
            'city': 'navsari',
            'country': 'India'
            };
            personalInfo['country'] = 'Canada';
            personalInfo['city'] = 'Toronto';
            print(personalInfo);


}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 5.dart
{name: rajan patel, profession: Computer Engineering Student, city: Toronto, country: Canada}

Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
```

3. Iterate over the map and print all the values.

```dart
void main(){

     Map<String, String> personalInfo = {
            'name': 'rajan patel',
            'profession': 'Computer Engineering Student',
            'city': 'navsari',
            'country': 'India'
            };
        for (var value in personalInfo.values) {
        print(value);
}
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 6.dart
rajan patel
Computer Engineering Student
navsari
India
```

Exercise:03

Given the following exam scores: final scores = [89, 77, 46, 93, 82, 67, 32, 88];

1. Use sort to find the highest and lowest grades.

```dart
void main(){

    List<int> finalScores = [89, 77, 46, 93, 82, 67, 32, 88];
    finalScores.sort();
    int lowestGrade = finalScores.first;
    int highestGrade = finalScores.last;
    print("Lowest Grade: $lowestGrade");
    print("Highest Grade: $highestGrade");
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 7.dart
Lowest Grade: 32
Highest Grade: 93
```

2. Use where to find all the B grades, that is, all the scores between 80 and 90.

```dart
void main(){
    List<int> finalScores = [89, 77, 46, 93, 82, 67, 32, 88];
    List<int> bGrades =
    finalScores.where((score) => score >= 80 && score <=
    90).toList();
    print("B Grades: $bGrades");

}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Co
$ dart 8.dart
B Grades: [89, 82, 88]
```

Challenge 1: A unique request Write a function that takes a paragraph of text and returns a collection of unique String characters that the text contains.

```dart
Set<String> getUniqueCharacters(String text) {
    Set<String> uniqueCharacters = {};
    for (var character in text.split('')) {
        if (character.trim().isNotEmpty) {
            uniqueCharacters.add(character);
        }
    }
    return uniqueCharacters;
}
void main(List<String> arguments) {
    String paragraph = "This is a paragraph of text.";
    Set<String> uniqueCharacters = getUniqueCharacters(paragraph);
    print("Unique Characters: $uniqueCharacters");
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 9.dart
Unique Characters: {T, h, i, s, a, p, r, g, o, f, t, e, x, .}
```

Challenge 2: Counting on you Repeat Challenge 1, but this time have the function return a collection that contains the frequency, or count, of every unique character.

```dart
Map<String, int> countCharacterFrequency(String text) {
```

```
    Map<String, int> characterFrequency = {};
    for (var character in text.split('')) {
        if (character.trim().isNotEmpty) {
            characterFrequency[character] =
            (characterFrequency[character] ?? 0) + 1;
        }
    }
    return characterFrequency;
}
void main(List<String> arguments) {
    String paragraph = "This is a paragraph of text.";
    Map<String, int> characterFrequency =
    countCharacterFrequency(paragraph);
    print("Character Frequency: $characterFrequency");
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 10.dart
Character Frequency: {T: 1, h: 2, i: 2, s: 2, a: 4, p: 2, r: 2, g: 1, o: 1, f: 1, t: 2, e: 1, x: 1, .: 1}
```

Challenge 3: Mapping users Create a class called User with properties for id and name. Make a List with three users, specifying any appropriate names and IDs you like. Then write a function that converts your user list to a list of maps whose keys are id and name.

```
class User {
  int id;
  String name;

  User(this.id, this.name);
}

List<Map<String, dynamic>> convertUsersToListOfMaps(List<User>
users) {
  List<Map<String, dynamic>> userMaps = [];
  for (var user in users) {
    Map<String, dynamic> userMap = {
      'id': user.id,
      'name': user.name,
    };
    userMaps.add(userMap);
  }
  return userMaps;
```

```dart
}

void main(List<String> arguments) {
  User user1 = User(1, 'rajan patel');
  User user2 = User(2, 'om Patel');
  User user3 = User(3, 'krishna Patel');
  List<User> usersList = [user1, user2, user3];
  List<Map<String, dynamic>> usersAsMaps =
convertUsersToListOfMaps(usersList);
  print(usersAsMaps);
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 11.dart
[{id: 1, name: rajan patel}, {id: 2, name: om Patel}, {id: 3, name: krishna Patel}]
```

Tutorial 4_2 :

Exercise : 01 :

1. Create a class named Fruit with a String field named color and a method named describeColor, which uses color to print a message.

```dart
class Fruit {
  String color;

  Fruit(this.color);

  void describeColor() {
    print("This fruit is $color in color.");
  }
 }

 void main(List<String> arguments) {
    Fruit apple = Fruit("red");
    Fruit banana = Fruit("yellow");

    apple.describeColor();
    banana.describeColor();
  }

```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64
$ dart 12.dart
This fruit is red in color.
This fruit is yellow in color.
```

2. Create a subclass of Fruit named Melon and then create two Melon subclasses named Watermelon and Cantaloupe.

```dart
class Fruit {
  String color;

  Fruit(this.color);

  void describeColor() {
    print("This fruit is $color in color.");
  }
}

class Melon extends Fruit {
  Melon(String color) : super(color);
}

class Watermelon extends Melon {
  Watermelon(String color) : super(color);
}

class Cantaloupe extends Melon {
  Cantaloupe(String color) : super(color);
}

void main(List<String> arguments) {
  Watermelon watermelon = Watermelon("green and red");
  Cantaloupe cantaloupe = Cantaloupe("orange");

  watermelon.describeColor();
  cantaloupe.describeColor();
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 13.dart
This fruit is green and red in color.
This fruit is orange in color.
```

3. Override describeColorin the Watermelon class to vary the output.

```dart
class Fruit {
  String color;

  Fruit(this.color);

  void describeColor() {
    print("This fruit is $color in color.");
  }
}

class Melon extends Fruit {
  Melon(String color) : super(color);
}

class Watermelon extends Melon {
  Watermelon(String color) : super(color);

  @override
  void describeColor() {
    print("This watermelon is mostly $color, with some green
stripes.");
  }
}

class Cantaloupe extends Melon {
  Cantaloupe(String color) : super(color);
}

void main() {
  Watermelon watermelon = Watermelon("green and red");
  Cantaloupe cantaloupe = Cantaloupe("orange");

  watermelon.describeColor();
  cantaloupe.describeColor();
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 14.dart
This watermelon is mostly green and red, with some green stripes.
This fruit is orange in color.

Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$
```

Exercise : 02 :

1. Create an interface called Bottle and add a method to it called open.

```dart
abstract class Bottle {
  void open();
}

class PlasticBottle extends Bottle {
  @override
  void open() {
    print("Twist the plastic cap to open the bottle.");
  }
}

class GlassBottle extends Bottle {
  @override
  void open() {
    print("Use a bottle opener to pop the metal cap and open the
glass bottle.");
  }
}

void main() {
  PlasticBottle plasticBottle = PlasticBottle();
  GlassBottle glassBottle = GlassBottle();

  plasticBottle.open();
  glassBottle.open();
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/flutter practice/lab4
$ dart 15.dart
Twist the plastic cap to open the bottle.
Use a bottle opener to pop the metal cap and open the glass bottle.
```

2. Create a concrete class called SodaBottle that implements Bottle and prints “Fizz fizz” when open is called.

```dart
abstract class Bottle {
  void open();
}

class SodaBottle implements Bottle {
  @override
  void open() {
    print("Fizz fizz");
  }
}

void main() {
  SodaBottle sodaBottle = SodaBottle();
  sodaBottle.open();
}
```

```
Rajankumar@LAPTOP-
$ dart 16.dart
Fizz fizz
```

3. Add a factory constructor to Bottle that returns a SodaBottle instance.

```dart
abstract class Bottle {
  void open();
  factory Bottle.sodaBottle() => SodaBottle();
}

class SodaBottle implements Bottle {
  @override
  void open() {
    print("Fizz fizz");
  }
```

```
}

void main() {
  Bottle bottle = Bottle.sodaBottle();
  bottle.open();
}
```

```
Rajankumar@LAPTOP-
$ dart 17.dart
Fizz fizz
```

4. Instantiate SodaBottle by using the Bottle factory constructor and call open on the object

```
abstract class Bottle {
  void open();
  factory Bottle.sodaBottle() => SodaBottle();
}

class SodaBottle implements Bottle {
  @override
  void open() {
    print("Fizz fizz");
  }
}

void main() {
  Bottle sodaBottle = Bottle.sodaBottle();
  sodaBottle.open();
}
```

```
$ dart 18.dart
Fizz fizz
```

Exercise : 03 :

1. Create a class called Calculator with a method called sum that prints the sum of any two integers you give it.

```
2. class Calculator {
3.   void sum(int a, int b) {
4.     int result = a + b;
5.     print("The sum of $a and $b is $result");
6.   }
```

```
7. }
8.
9. void main() {
10.       Calculator calculator = Calculator();
11.       calculator.sum(5, 7);
12.     }
13.
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d
$ dart 19.dart
The sum of 5 and 7 is 12
```

2. Extract the logic in sum to a mixin called Adder

```
mixin Adder {
  void sum(int a, int b) {
    int result = a + b;
    print("The sum of $a and $b is $result");
  }
}

class Calculator with Adder {
  // The rest of the class remains the same
}

void main() {
  Calculator calculator = Calculator();
  calculator.sum(5, 7);
}
```

```
Rajankumar@LAPTOP-5P5GTVER MIN
$ dart 19.dart
The sum of 5 and 7 is 12
```

3. Use the mixin in Calculator.

```
mixin Adder {
```

```
  void sum(int a, int b) {
    int result = a + b;
    print("The sum of $a and $b is $result");
  }
}

class Calculator with Adder {
  // The rest of the class remains the same
}

void main() {
  Calculator calculator = Calculator();
  calculator.sum(5, 7);
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /
$ dart 20.dart
The sum of 5 and 7 is 12
```

Challenges: Challenge 1: Heavy monotremes Dart has a class named Comparable, which is used by the the sort method of List to sort its elements. Add a weight field to the Platypus class you made in this lesson. Then make Platypus implement Comparable so that when you have a list of Platypus objects, calling sort on the list will sort them by weight.

```
class Platypus implements Comparable<Platypus> {
String name;
int age;
double weight;
Platypus(this.name, this.age, this.weight);
@override
int compareTo(Platypus other) {
return weight.compareTo(other.weight);
}
@override
String toString() {
return 'Platypus{name: $name, age: $age, weight: $weight}';
}
}
```

```
void main() {
List<Platypus> platypusList = [
Platypus('Kevin', 5, 2.5),
Platypus('Yash', 3, 1.8),
Platypus('Har', 7, 3.1),
];
platypusList.sort();
for (var platypus in platypusList) {
print(platypus);
}
}
```

```
Rajankumar@LAPTOP-5P5GTVER MINGW64 /d/Collage/
$ dart 21.dart
Platypus{name: Yash, age: 3, weight: 1.8}
Platypus{name: Kevin, age: 5, weight: 2.5}
Platypus{name: Har, age: 7, weight: 3.1}
```

Challenge 2: Fake notes Design an interface to sit between the business logic of your note-taking app and a SQL database. After that, implement a fake database class that will return mock data.

Part 1: Designing an Interface

```
abstract class NoteDatabase {
  Future<void> open();
  Future<void> close();
  Future<List<Note>> getAllNotes();
  Future<void> insertNote(Note note);
  Future<void> updateNote(Note note);
  Future<void> deleteNote(Note note);
}

class Note {
  final int id;
  final String title;
  final String content;

  Note({
    required this.id,
    required this.title,
```

```
    required this.content,
  });
}
```

Part 2: Implementing a Fake Database

```
class FakeNoteDatabase implements NoteDatabase {
  final List<Note> _notes = [];

  @override
  Future<void> open() async {
    // Simulate opening the database
    print("Fake database opened");
  }

  @override
  Future<void> close() async {
    // Simulate closing the database
    print("Fake database closed");
  }

  @override
  Future<List<Note>> getAllNotes() async {
    return _notes;
  }

  @override
  Future<void> insertNote(Note note) async {
    _notes.add(note);
  }

  @override
  Future<void> updateNote(Note note) async {
    int index = _notes.indexWhere((n) => n.id == note.id);
    if (index != -1) {
      _notes[index] = note;
    }
  }

  @override
```

```
  Future<void> deleteNote(Note note) async {
    _notes.removeWhere((n) => n.id == note.id);
  }
}

void main() async {
  final database = FakeNoteDatabase();
  await database.open();
  await database.insertNote(Note(id: 1, title: "Note 1", content:
"Content 1"));
  await database.insertNote(Note(id: 2, title: "Note 2", content:
"Content 2"));
  List<Note> notes = await database.getAllNotes();
  print(notes);
  await database.close();
}
```

```
Fake database opened
[Instance of 'Note', Instance of 'Note']
Fake database closed
```

Challenge 3: Time to code Dart has a Duration class for expressing lengths of time. Make an extension on int so that you can express a duration like so: final timeRemaining = 3.minutes;

```
extension DurationExtension on int {
  Duration get milliseconds => Duration(milliseconds: this);
  Duration get seconds => Duration(seconds: this);
  Duration get minutes => Duration(minutes: this);
  Duration get hours => Duration(hours: this);
  Duration get days => Duration(days: this);
}

void main() {
  final timeRemaining = 3.minutes;
  print("Time remaining: $timeRemaining");
}
```

```
Time remaining: 0:03:00.000000
```