This note explains Functions, Collections, and Lambdas in a simple, clear way, with examples.

1. Functions

A function is a block of code that performs a specific task. You can think of it like a recipe: you give it some inputs (ingredients), it does something, and sometimes it gives you a result (like a cake).

Key Points:

- Functions in Dart are defined using a return type, name, and body.
- They can take parameters (inputs) and return a value (output).
- If no return type is needed, use void.

Example

```dart
// A simple function that prints a greeting
void sayHello(String name) {
  print('Hello, $name!');
}

// A function that returns a value
int add(int a, int b) {
  return a + b;
}

void main() {
  sayHello('Alice'); // Output: Hello, Alice!
  int result = add(3, 5); // Calls add function, returns 8
  print(result); // Output: 8
}
```

Explanation:

- sayHello takes a parameter (name) and prints a message but doesn't return anything (void).
- add takes two parameters (a and b), adds them, and returns the result (int).

See functions as reusable tools. In the sense that, instead of writing the same code multiple times, you put it in a function and call it whenever needed.

Parameters can be viewed as placeholders for values you provide when calling the function.

While “return” gives back a result to the caller (like handing over the cake after baking).

2. Collections

Collections are ways to store multiple pieces of data in one place, like a bag holding toys. Dart has three main types of collections: Lists, Maps, and Sets.

a. Lists
A List is an ordered collection of items, like a shopping list. You can access items by their position (index), starting from 0.

```dart
void main() {
  // Creating a list
  List<String> fruits = ['Apple', 'Banana', 'Orange'];

  // Accessing items
  print(fruits[0]); // Output: Apple

  // Adding an item
  fruits.add('Mango');
  print(fruits); // Output: [Apple, Banana, Orange, Mango]

  // Length of the list
  print(fruits.length); // Output: 4
}
```

Key Points

- Lists are ordered, so fruits[0] is always the first item.
- You can add, remove, or change items.
- Use List<Type> to specify what kind of data the list holds (e.g., String, int – see my previous note “Dart Basic” to understand built-in types in Dart).

b. Maps
A Map stores data as key-value pairs, like a dictionary where you look up a word (key) to find its meaning (value).

```dart
void main() {
  // Creating a map
  Map<String, int> ages = {
    'Alice': 25,
    'Bob': 30,
    'Charlie': 35,
  };

  // Accessing a value
  print(ages['Alice']); // Output: 25

  // Adding a new key-value pair
  ages['David'] = 40;
  print(ages); // Output: {Alice: 25, Bob: 30, Charlie: 35, David: 40}
}
```

Key Points

- Keys are unique (e.g., 'Alice' can only appear once).
- You access values using the key, like ages['Alice'].
- Maps are unordered, so the order of key-value pairs doesn't matter.

c. Sets

A Set is a collection of unique items, like a bag of coins where no two coins are the same.

```dart
void main() {
  // Creating a set
  Set<String> colors = {'Red', 'Blue', 'Green'};

  // Adding an item
  colors.add('Yellow');
  print(colors); // Output: {Red, Blue, Green, Yellow}

  // Adding a duplicate (won't be added)
  colors.add('Red');
  print(colors); // Output: {Red, Blue, Green, Yellow} (no duplicate Red)
}
```

Key points

- Sets don't allow duplicates.
- Sets are unordered, so the order of items isn't guaranteed.
- Useful when you need unique items only.

Please note that “Lists” are like numbered slots for items (e.g., a playlist of songs). While “Maps” are like labeled boxes where each label (key) points to something (value) – each key must be unique. And “Sets” are like a unique collection of items with no repeats.

3. Lambdas
   A lambda (also called an anonymous function or closure) is a small, unnamed function used for quick tasks. You can think of it as a one-line shortcut for a function.

   Lambdas are often used in collections for operations like filtering or transforming data. They use the “=>” (fat arrow) syntax for concise code.

```dart
void main() {
  List<int> numbers = [1, 2, 3, 4];

  // Using a lambda to double each number
  var doubled = numbers.map((number) => number * 2).toList();
  print(doubled); // Output: [2, 4, 6, 8]

  // Lambda to filter even numbers
  var evens = numbers.where((number) => number % 2 == 0).toList();
  print(evens); // Output: [2, 4]
}
```

   Observing the above code, in map, the lambda (number) => number * 2 takes each number and doubles it. While in where, the lambda (number) => number % 2 == 0 checks if a number is even.

   Lambdas are like mini functions you write directly where they’re needed. They’re often used with collections to process data quickly (e.g., map to transform, where to filter). The => means “return this result.” For example, number => number * 2 means “take a number and return it doubled.”

   To conclude this note, let’s look at an example combining functions, collections, and lambdas:

Summary

Functions are reusable blocks of code. They can take inputs (parameters) and give outputs (returns). For collections we have Lists, Maps and Sets. Lists are ordered and can be

viewed like a numbered list of items. Maps are key-value pairs (like a dictionary), While Sets are unique items with no duplicates. Also, lambdas are short, unnamed functions (using fat arrow “=>”) for quick tasks, and are often used with collections.