

# Week 01: Dart Fundamentals Documentation

---

## Executive Summary

This document provides comprehensive technical documentation for Week 01 deliverables of the NeuroApp Flutter Internship Program. The implementation demonstrates mastery of Dart programming fundamentals through four systematically developed modules covering core syntax, data structures, object-oriented programming, and asynchronous operations.

## Objectives:

- Dart Setup & Basic Syntax
- Dart Collections & Higher-Order Functions
- Object-Oriented Programming in Dart
- Asynchronous Programming

## Module 1: Dart Basics

1. **var** A way to declare a variable without specifying a type; the type is inferred from the initial value. The value **can be changed** later, provided it stays the same data type.
2. **final** Used for a variable that can only be set **once**; it is initialized the first time it is accessed. It is a **runtime** constant, meaning its value can be determined while the app is running (like the current time).
3. **const** a **compile-time** constant that must be initialized with a value known before the program runs. It is implicitly final, but even more restrictive: the value and everything it contains must be immutable.
4. **int**: Represents **whole numbers** (64-bit integers) without decimals, such as 10 or -500.
5. **double**: Represents **fractional numbers** (64-bit floating-point) for precision, such as 3.14 or 0.001.
6. **String**: Represents **textual data** as a sequence of UTF-16 code units, wrapped in quotes.
7. **bool**: Represents **logical values** used in conditional branching, restricted to true or false.
8. **If-Else Conditions** Executes specific code blocks based on a Boolean evaluation to manage program branching.
9. **For & While Loops** Iterate over a range or collection using for known counts and while for condition-based execution.
10. **Sum Function** A standard arithmetic function that accepts two integer parameters and returns their mathematical sum.
11. **Recursive Factorial** A function that calculates  $n!$  By calling itself until it reaches the base case of  $n \leq 1$ .
12. **Is Prime Function** Determines if a number is a prime by verifying it has no divisors other than 1 and itself?

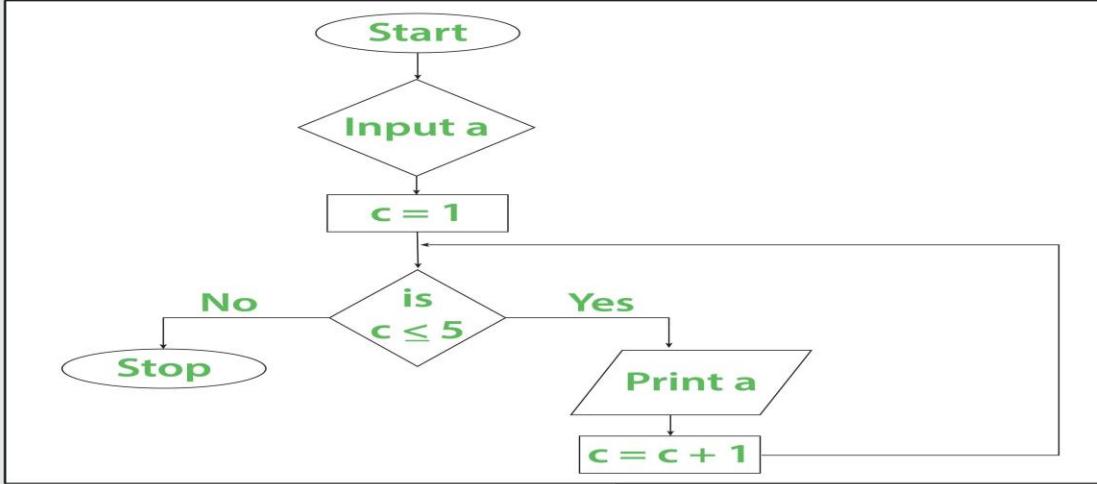


Figure 1 logic for finding prime Number

## Module 2: Collections Framework

1. **List Operations** A dynamic array used to store ordered objects; supports add (value), insert (index, value), and remove (value).
2. **Set (Unique Elements)** an unordered collection of unique items that automatically prevents duplicate entries.
3. **Map (Key-Value Pairs)** a dictionary-style collection that associates unique keys with specific values for fast retrieval.
4. **list.map () Transformation** Applies a function to each element in a list to produce a new collection of transformed data.
5. **list.where () Filtering** Returns a lazy iterable of all elements that satisfy a specific Boolean predicate.
6. **list.reduce() Aggregation** Combines all elements of a collection into a single value using a provided binary function.
7. **List of Maps Structure** A complex data structure used to represent a collection of objects (students) with multiple attributes.
8. **Add Student Records** Populating the collection with multiple map entries containing various data types like String and int.
9. **Sort by Marks Descending** Utilizes the sort method with comparison logic to arrange records from highest to lowest score.
10. **Filter High Achievers (Marks > 75)** Uses the where method to isolate records of students meeting a specific academic threshold.
11. **Search by Name** Employs the first Where method to retrieve the first specific record matching a name string.
12. **Formatted Output** Iterates through the collection to print clean, readable strings using string interpolation.

## Module 3: Object-Oriented Programming

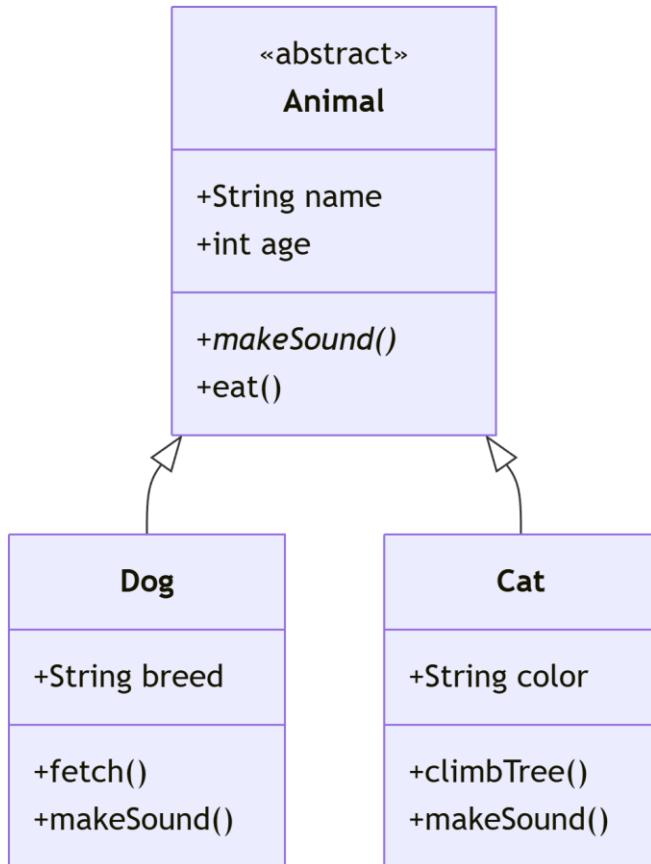
1. **User Class & Private Members** Encapsulation data using the underscore prefix to ensure variables are library-private.
2. **Named Constructor** Provides an alternative way to instantiate a class with specific parameters for better readability.
3. **Getters and Setters** Mediates access to private properties, allowing for validation or logic during data retrieval and updates.
4. **Admin Extension** Demonstrates inheritance by allowing the Admin subclass to inherit properties and behaviors from User.
5. **Method Overriding** Redefines the `toString()` method to provide a custom, human-readable string representation of an object.
6. **Abstract Animal Class** Defines a blueprint for other classes that cannot be instantiated directly, enforcing a shared structure. Implements the Animal blueprint in Dog and Cat classes to provide specific logic for each type.
7. **Abstract Method** Forces subclasses to provide a specific implementation for a method, ensuring consistent behavior across types.
8. **Product Model** Defines a simple data model with immutable or mutable properties to represent a physical item.
9. **Shopping Cart Manage** Utilizes a collection-based class to manage a group of Product objects through composition.
10. **Cart Operations** Implements business logic to manipulate the collection and calculate aggregate data.
11. **Execution in main ()** the entry point of the application where objects are instantiated and methods are called to verify logic.

## Module 4: Asynchronous Programming

1. **Future Delayed Response** Represents a potential value or error that will be available at a later time, often used for simulating network latency.
2. **Async-Await Execution** A syntax pattern that pauses function execution until a Future completes, providing a linear way to handle non-blocking operations.
3. **Try-Catch Error Handling** The standard mechanism for intercepting and managing exceptions that occur during the lifecycle of an asynchronous task.
4. **Multiple Future Functions** The practice of modularizing distinct asynchronous tasks into separate, reusable logic units for better maintainability.
5. **Future.wait() Parallelism** A coordination pattern that executes multiple independent operations concurrently and returns their results once all tasks finish.
6. **Stream Generator Function** A specialized producer that emits a continuous sequence of data events over time rather than a single discrete value.
7. **Stream Consumption (await for)** an asynchronous loop that listens to a sequence of events and processes each item as it is pushed into the stream.
8. **Stream Error Handling** The implementation of specific listeners or recovery logic to manage interruptions or failures within a data pipeline.
9. **Stream Controller Management** A low-level primitive that acts as both a source and a controller, allowing developers to manually inject data into a stream.

**10. Stream Transformation** The process of intercepting raw stream data to filter, modify, or combine events before they reach the subscriber.

## Class Hierarchy



**Document Prepared By:** Youmna Saifullah

**Date:** 26 December 2025

**Internship Program:** NeuroApp

