**1. Fundamental Data Types in Dart and Their Uses**

Dart supports various data types to store different types of values. These data types help in defining variables, processing data, and performing operations efficiently.

**1.1 int (Integer)**

* Represents whole numbers without decimals.
* Used for counting, indexing, and mathematical calculations.
* Example:

int age = 25;

print("Your age is $age"); // Output: Your age is 25

**1.2 double (Floating Point Number)**

* Represents numbers with decimal points.
* Useful for precise calculations like price, weight, and percentages.
* Example:

double price = 99.99;

print("Total price is $price"); // Output: Total price is 99.99

**1.3 String (Text Data)**

* Stores a sequence of characters enclosed in single or double quotes.
* Used for names, messages, and text-based data.
* Example:

String name = "Karan";

print("Hello, $name!"); // Output: Hello, Karan!

**1.4 bool (Boolean)**

* Stores true or false values.
* Used in decision-making conditions and logical operations.
* Example:

bool isFlutterAwesome = true;

print("Is Flutter amazing? $isFlutterAwesome"); // Output: Is Flutter amazing? true

**1.5 List (Collection of Values - Array)**

* Stores multiple values of the same or different types.
* Indexed from 0 (first element) to n-1 (last element).
* Example:

List<String> fruits = ["Apple", "Banana", "Mango"];

print(fruits[1]); // Output: Banana

**1.6 Map (Key-Value Pair Collection - Dictionary)**

* Stores data as key-value pairs for fast lookup.
* Example:

Map<String, int> studentScores = {"Alice": 90, "Bob": 85, "Charlie": 95};

print(studentScores["Bob"]); // Output: 85

**1.7 var (Type Inference Variable)**

* Detects the type automatically based on the assigned value.
* Once assigned, it behaves like a strongly-typed variable.
* Example:

var x = 10; // x is inferred as int

print(x);

**1.8 dynamic (Flexible Data Type)**

* Can hold any type of data, and its type can change during runtime.
* Example:

dynamic y = "Hello";

y = 20; // No error, y is now an int

print(y);

**2. Control Structures in Dart**

Control structures in Dart help manage the execution flow of a program based on conditions or loops.

**2.1 if Statement (Conditional Execution)**

* Executes a block of code if the condition is true.
* Example:

int number = 10;

if (number > 0) {

print("Positive number");

}

**2.2 if-else Statement (Two-way Condition)**

* Executes one block if the condition is true, otherwise executes another.
* Example:

int score = 45;

if (score >= 50) {

print("Pass");

} else {

print("Fail");

}

**2.3 for Loop (Fixed Iteration)**

* Repeats a block of code for a known number of times.
* Example:

for (int i = 1; i <= 5; i++) {

print("Iteration $i");

}

**2.4 while Loop (Conditional Repetition)**

* Executes a block repeatedly while a condition is true.
* Example:

int count = 1;

while (count <= 3) {

print("Count: $count");

count++;

}

**2.5 do-while Loop (Ensures Execution At Least Once)**

* Executes once before checking the condition.
* Example:

int num = 1;

do {

print("Hello, Dart!");

num++;

} while (num <= 3);

**2.6 switch Statement (Multiple Cases Condition)**

* Used when checking multiple values of a variable.
* Example:

String grade = "A";

switch (grade) {

case "A":

print("Excellent!");

break;

case "B":

print("Good Job!");

break;

default:

print("Keep Trying!");

}

**3. Object-Oriented Programming (OOP) in Dart**

Dart follows OOP principles, which make code reusable and organized.

**3.1 Class and Object**

* A class is a blueprint for objects.
* Objects are instances of a class.
* Example:

class Car {

String brand = "Toyota";

void display() {

print("Car Brand: $brand");

}

}

Car myCar = Car();

myCar.display();

**3.2 Inheritance (Extending a Class)**

* Allows one class to acquire properties of another.
* Example:

class Animal {

void eat() {

print("Eating...");

}

}

class Dog extends Animal {

void bark() {

print("Barking...");

}

}

Dog myDog = Dog();

myDog.eat();

myDog.bark();

**3.3 Polymorphism (Method Overriding)**

* A subclass can redefine a method from its parent class.
* Example:

class Animal {

void sound() {

print("Animal makes a sound");

}

}

class Cat extends Animal {

@override

void sound() {

print("Meow!");

}

}

Cat myCat = Cat();

myCat.sound();

**3.4 Interfaces (implements)**

* Used to enforce a structure in a class.
* Example:

class Printer {

void printData();

}

class LaserPrinter implements Printer {

@override

void printData() {

print("Printing...");

}

}

LaserPrinter lp = LaserPrinter();

lp.printData();

**4. Asynchronous Programming in Dart**

Asynchronous programming is used for tasks that take time, like fetching data or network requests.

**4.1 Future (Delayed Data Handling)**

* Represents a value that will be available later.
* Example:

Future<String> fetchData() async {

return Future.delayed(Duration(seconds: 2), () => "Data Loaded");

}

void main() async {

print("Fetching...");

String data = await fetchData();

print(data);

}

**4.2 async and await (Non-Blocking Code Execution)**

* Used to handle Futures in a readable way.

**4.3 Stream (Handling Continuous Data Flow)**

* Used for receiving multiple values over time, like user inputs or sensor data.