Dart&Flutter concepts

1️⃣ Dart Basics

🔹 Lambda (Arrow) Functions

- Represent short functions using `=>`.
- Often used with **callbacks**.
- Can be **named or anonymous**.
- Must contain **only one expression** after `=>`.

```
intsquare(intx)=>x*x;

vargreet= (Stringname)=>"Hello$name";
```

If multiple statements are needed, use a block function:

```
intsum(inta,intb) {
returna+b;
}
```

🔹 Higher-Order Functions (HOF)

A function that:

- Takes another function as a parameter, OR
- Returns a function.

```
voidexecute(Functionaction) {
action();
}
```

Example with List:

```
varnumbers= [1,2,3];
numbers.forEach((n)=>print(n));
```

🔹 `where()` Method

- A **higher-order function** on `Iterable`.
- Works like `filter()` in other languages.
- Returns an **Iterable**, NOT a tuple.

```
varnumbers= [1,2,3,4];
vareven=numbers.where((n)=>n%2==0);
print(even);// (2, 4)
```

🔹 Iterable

An `Iterable` is a collection that can be looped over.

Examples:

- `List`
- `Set`
- `Map` (iterates over keys by default)
- `String` (iterable of characters)

```
Iterable<int>nums= [1,2,3];
```

🔹 Lexical Closure (غلق معجمي)

A function defined inside another function that:

- Accesses outer variables
- Remembers them even after outer function ends

```
Functioncounter() {
intcount=0;
return () {
count++;
returncount;
  };
}
```

🔹 Extension Methods

Add methods to existing classes without modifying them.

```
extensionStringExtensiononString {
Stringshout()=>toUpperCase()+"!";
}
```

🔹 Platform Class

From `dart:io`.

```
import'dart:io';

if (Platform.isWindows) {
```

```
print("Running on Windows");
}
```

Used to detect operating system.

2️⃣ Object-Oriented Programming (OOP)

🔹 Everything is an Object

- `int`, `double`, `String`, `List` — all are objects.
- Every class implicitly extends `Object`.

Important `Object` members:

- `runtimeType`
- `hashCode`
- `toString()`
- `noSuchMethod()`

🔹 Abstract Class

- Cannot create objects directly.
- May contain abstract methods (without body).

```
abstractclassAnimal {
voidmakeSound();
}
```

A class that contains at least one abstract method **must be abstract**.

🔹 Inheritance (`extends`)

- Dart supports **single inheritance only**.

```
classDogextendsAnimal {
@override
voidmakeSound()=>print("Bark");
}
```

🔹 Interfaces (`implements`)

Dart does not have a separate `interface` keyword.

Any class can act as an interface using `implements`.

```
classRobotimplementsAnimal {
@override
voidmakeSound() {
print("Beep");
  }
}
```

Important:

- Must override **all methods and properties**
- Even implemented ones

Supports multiple interfaces:

```
classMyClassimplementsA,B {}
```

🔹 Mixins (`with`)

Used to reuse behavior across multiple classes.

```
mixinFly {
voidfly()=>print("Flying");
}
```

```
classBirdwithFly {}
```

- Enables multiple behavior reuse
- Not true multiple inheritance

🔹 Polymorphism

Greek meaning: *Many forms*.

A subclass can be treated as its superclass.

```
Animala=Dog();
a.makeSound();
```

In Dart, polymorphism is achieved through:

- Method overriding (not overloading)

🔹 Method Overriding

Subclass changes behavior of parent method.

```
@Override
voidmakeSound() {
print("Different sound");
}
```

🔹 Named Constructors

Since Dart does not support function overloading:

```
classPerson {
Stringname;

Person(this.name);
Person.guest() :name="Guest";
}
```

🔹 Optional Parameters

Positional

```
voidgreet([Stringname="Guest"]) {}
```

Named

```
voidgreet({Stringname="Guest"}) {}
```

🔹 Enums

Used for fixed set of constants.

```
enumStatus {loading,success,error }
```

Prevents using raw strings.

🔹 Generics

Allow class to work with any data type.

```
classBox<T> {
Tvalue;
Box(this.value);
}
```

Type is decided at compile time (not runtime).

🔹 Built-in Methods Example

```
print(12.gcd(8));// 4
```

🔹 Creating External Packages

You can:

- Create reusable Dart files
- Publish packages
- Import using:

```
import'package:my_package/my_file.dart';
```

3️⃣ Flutter Setup

🔹 Installation Steps

1. Download Flutter SDK (zip).
2. Extract to `C:\flutter`
    - Avoid:
        - Program Files
        - Windows folder
        - Downloads
3. Add `flutter/bin` to **Environment Variables (PATH)**.
4. Run:

```
flutter--version
```

🔹 Install Tools

- Android Studio
- VS Code
- Dart & Flutter extensions
- AVD (Android Virtual Device)
- SDK platforms & tools

🔹 Useful Commands

```
flutter doctor
flutter create app_name
dart run file.dart
```

🔹 VS Code Extension

- **Error Lens** → Shows errors inline while typing.