Function in Dart

1️⃣ What is a Function?

A **function** is a reusable block of code that performs a specific task.

Why Functions?

- Avoid code repetition
- Improve readability
- Make programs modular and testable

2️⃣ Basic Function Syntax

```
returnType functionName(parameters) {
  // function body
  return value;
}
```

Example

```
int add(int a, int b) {
  return a + b;
}
```

3️⃣ Calling a Function

```
void main() {
  int result = add(3, 4);
  print(result); // 7
}
```

4️⃣ Types of Functions in Dart

Functions can be classified based on **parameters** and **return type**.

4.1 No Parameter and No Return Type

These functions do not take any input and do not return any value. They are mainly used for **displaying output or performing actions**.

```dart
void greet() {
  print("Hello, Welcome to Dart!");
}
```

4.2 Parameter and No Return Type

These functions take input parameters but do not return any value. They are commonly used for **processing data and printing results**.

```dart
void printSum(int a, int b) {
  print(a + b);
}
```

4.3 No Parameter and Return Type

These functions do not take any parameters but return a value. They are useful when a function needs to **compute and provide a result**.

```dart
int getNumber() {
  return 10;
}
```

4.4 Parameter and Return Type

These functions take parameters and return a value. This is the **most commonly used type** of function.

```dart
int add(int a, int b) {
  return a + b;
}
```

5️⃣Parameters in Dart Functions

We can define **optional parameters** in Dart in two ways:

- Optional positional parameters
- Optional named parameters

These parameters must always appear **after required parameters** in a function's signature.

5.1 Optional Positional Parameters

Optional positional parameters are enclosed in **square brackets []**. If the caller does not provide a value, the parameter defaults to `null` unless a default value is specified.

Syntax

```
void sayHello(String name, [String? title]) {
  if (title != null) {
    print('Hello $title $name!');
  } else {
    print('Hello $name!');
  }
}
```

Usage

```
sayHello('Bishal');             // Output: Hello Bishal!
sayHello('Vivek', 'Professor'); // Output: Hello Professor Vivek!
```

With Default Values

You can provide a default value to avoid null checks.

```
void sayHello(String name, [String title = 'Guest']) {
  print('Hello $title $name!');
}

// Usage
sayHello('Vivek'); // Output: Hello Guest vivek!
```

5.2 Optional Named Parameters

Optional named parameters are enclosed in **curly braces {}**. They are optional by default and can be passed in any order using their parameter names.

Syntax

```
void greetUser({String? greeting, String? name}) {
  print('${greeting ?? "Hello"}, ${name ?? "Stranger"}!');
}
```

Usage

```
greetUser(name: 'Vivek', greeting: 'Welcome'); // Output: Welcome, Vivek!
greetUser(greeting: 'Good morning');          // Output: Good morning, Stranger!
greetUser();                                  // Output: Hello, Stranger!
```

With Default Values and `required` Keyword

You can assign default values using the `=` operator. To make a named parameter mandatory, use the `required` keyword.

```
void greet({required String name, String title = 'Guest'}) {
  print('Hello $title $name!');
}

// Usage
greet(name: 'Alice');              // Output: Hello Guest Alice!
greet(name: 'Bob', title: 'Dr.'); // Output: Hello Dr. Bob!
// greet(title: 'Dr.');            // Error: A value for 'name' must be provided

---
```

6️⃣Arrow (Fat Arrow) Functions

Arrow functions are used for **single-expression functions**. They provide a shorter and cleaner syntax.

Syntax

```
returnType functionName(parameters) => expression;
```

Example

```
int multiply(int a, int b) => a * b;
```

Arrow functions automatically return the expression value and are commonly used in **Collection callbacks**, `map`, `where`, and other functional operations.

7️⃣Functions as First-Class Citizens

In Dart, functions are **first-class citizens**, which means:

- Functions can be assigned to variables
- Functions can be passed as arguments to other functions
- Functions can be returned from other functions

7.1 Assign Function to Variable

```
int add(int a, int b) {
  return a + b;
}

void main() {
  var operation = add; // function assigned to variable
  print(operation(2, 3)); // 5
}
```

7.2 Pass Function as Parameter

```
int add(int a, int b) => a + b;
int multiply(int a, int b) => a * b;

void calculate(int a, int b, int Function(int, int) op) {
  print(op(a, b));
}
```

```
void main() {
  calculate(2, 3, add);      // 5
  calculate(2, 3, multiply); // 6
}
```

7.3 Return Function from Another Function

```
Function makeMultiplier(int factor) {
  return (int value) => value * factor;
}

void main() {
  var doubleIt = makeMultiplier(2);
  print(doubleIt(5)); // 10
}
```

8️⃣Anonymous (Lambda) Functions

(Lambda) Functions

A function **without a name**.

```
var square = (int x) => x * x;
```

Common Usage

```
list.forEach((item) {
  print(item);
});
```

9️⃣Higher-Order Functions (HOF)

A function that **takes or returns another function**.

```
void execute(Function task) {
  task();
}
```

Built-in HOFs:

- map
- where
- reduce
- fold

1️⃣0️⃣ Closures (VERY IMPORTANT)

A **closure** is a function that remembers variables from its outer scope.

```
Function counter() {
  int count = 0;
  return () => ++count;
}
```

Each closure has its **own memory**.

1️⃣1️⃣ Best Practices

- Keep functions small
- Prefer pure functions
- Avoid side effects
- Use named parameters
- Avoid deeply nested functions

1️⃣2️⃣ Common Mistakes

❌Forgetting return ❌Calling function instead of passing it ❌Heavy logic inside UI code

---

1️⃣3️⃣ Interview Questions to Practice

1. What are first-class functions?
2. What is a closure?
3. map vs for loop?

✅Summary

- Functions are the backbone of Dart & Flutter

- Flutter UI is built using functions
- Closures and callbacks are used everywhere

☸️Next Topics Suggested

- Async & Await in depth
- Functional programming for Flutter
- State management using functions
- Widget lifecycle & callbacks