# Functions in Dart

A **function** is a **block of reusable code** that performs a specific task.
Instead of writing the same code multiple times, we can **define it once** and **call it whenever needed**.

## Syntax of a Function

```
returnType functionName(parameters) {
  // code block
  return value;
}
```

### Example:

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

### Explanation:

- `void` → means this function does **not return** any value.
- `greet` → function name.
- `{ }` → block of code to execute when the function is called.

### Calling the Function:

```
void main() {
  greet(); // function call
}
```

### Output:

```
Welcome to Dart!
```

## Types of Functions in Dart

There are mainly **four types of functions** based on **parameters** and **return type**.

| Parameters | Return Value | Example |
|---|---|---|
| Without Parameters, Without Return | | `void greet()` |
| With Parameters, Without Return | params, return | `void display(String name)` |
| Without Parameters, With Return | params, return | `int getNumber()` |
| With Parameters, With Return | params, return | `int add(int a, int b)` |

Let's understand each type clearly

## 1. Function without Parameters and without Return Type

**Example:**

```
void sayHello() {
  print("Hello User!");
}

void main() {
  sayHello();
}
```

**Output:**

```
Hello User!
```

**Explanation:**
No parameters and no value is returned. It only performs an action.

## 2. Function with Parameters and without Return Type

**Example:**

```
void display(String name, int age) {
  print("Name: $name");
  print("Age: $age");
}

void main() {
  display("Neeraj", 23);
}
```

**Output:**

```
Name: Neeraj
Age: 23
```

**Explanation:**
Here, we **pass values** (arguments) while calling the function.

## 3. Function without Parameters but with Return Type

**Example:**

```
int getLuckyNumber() {
  return 7;
}

void main() {
  int number = getLuckyNumber();
  print("Your lucky number is $number");
}
```

**Output:**

```
Your lucky number is 7
```

**Explanation:**
No parameter is passed, but the function **returns** a value.

### 4. Function with Parameters and with Return Type

**Example:**

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

void main() {
  int result = add(10, 20);
  print("Sum: $result");
}
```

**Output:**

```
Sum: 30
```

**Explanation:**
This is the most common type of function — it takes inputs and returns an output.

### Why Use Functions?

Improves **code reusability**
Increases **readability**
Easier to **test and maintain**
Helps in **modular programming**

## Parameters in Dart Functions

Dart supports **different types of parameters** for flexibility.

- **Positional parameters**
- **Optional parameters**
- **Named parameters**
- **Default parameter values**
- **Anonymous functions**
- **Arrow functions (short-hand syntax)**

In Dart, functions can have **different types of parameters** depending on how you want to pass values.

### 1. Required Parameters

These are **must** be passed when calling a function.

```
void greet(String name) {
  print("Hello, $name!");
}
```

```
void main() {
  greet("Neeraj"); // Works
  // greet(); Error: Missing argument
}
```

## 2. Optional Positional Parameters

Use **square brackets `[ ]`**.
If you don't pass a value, Dart uses `null` or a default value (if provided).

```
void greet(String name, [String? message]) {
  print("Hello, $name!");
  if (message != null) print(message);
}

void main() {
  greet("Neeraj"); // Only name
  greet("Neeraj", "Welcome back!"); // Both
}
```

## 3. Optional Named Parameters

Use **curly braces `{ }`** — you can pass parameters by **name**, in any order.

```
void showInfo({String? name, int? age}) {
  print("Name: $name, Age: $age");
}

void main() {
  showInfo(name: "Neeraj", age: 23);
  showInfo(age: 25, name: "Ravi"); // Order doesn't matter
}
```

## 4. Required Named Parameters

Add **`required`** keyword to make named parameters mandatory.

```
void showDetails({required String name, required int age}) {
  print("Name: $name, Age: $age");
}

void main() {
  showDetails(name: "Neeraj", age: 23); //
  // showDetails(name: "Neeraj"); Error: missing age
}
```

## 5. Default Parameter Values

You can set default values so if no argument is passed, Dart uses that default.

```
void greet(String name, {String message = "Welcome!"}) {
  print("Hello, $name! $message");
}
```

```
void main() {
  greet("Neeraj"); // Uses default message
  greet("Neeraj", message: "Good Morning!"); // Custom message
}
```

## 6. Combining Types

You can mix positional + named parameters.

```
void registerUser(String username, {int age = 18, String country =
"India"}) {
  print("Username: $username, Age: $age, Country: $country");
}

void main() {
  registerUser("Neeraj");
  registerUser("Ravi", age: 21, country: "USA");
}
```

## Anonymous & Arrow Functions in Dart

An **anonymous function** (also called **lambda** or **closure**) is a function **without a name**.
It can be **stored in a variable**, **passed as an argument**, or **used inline**.

**Syntax:**

```
(parameters) {
  // code
};
```

## Example 1 — Assigning an Anonymous Function to a Variable

```
void main() {
  var greet = (String name) {
    print("Hello, $name!");
  };

  greet("Neeraj"); // Output: Hello, Neeraj!
}
```

Here:

- `greet` is a variable that stores an **unnamed function**.
- You can call it just like a normal function.

## Example 2 — Passing Anonymous Function as an Argument

```
void performAction(Function action) {
  action();
}

void main() {
  performAction(() {
    print("This is an anonymous function!");
  });
}
```

Anonymous functions are often used in Flutter widgets, like in buttons:

```
ElevatedButton(
  onPressed: () {
    print("Button Clicked!");
  },
  child: Text("Click Me"),
);
```

## Arrow Functions (Short-hand Syntax)

Arrow functions are **a shorter way** to write simple one-line functions.
They use the **=>** symbol instead of `{ }`.

### Syntax:

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

The `=>` means *"return this expression"* automatically.

### Example 1 — Simple Arrow Function

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

void main() {
  print(add(5, 3)); // Output: 8
}
```

This is same as:

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

### Example 2 — Arrow Function in Variable

```
void main() {
  var greet = (String name) => print("Hello, $name!");
  greet("Neeraj"); // Output: Hello, Neeraj!
}
```

### Example 3 — Using Arrow Function with List Methods

```
void main() {
  var numbers = [1, 2, 3, 4];
  var squared = numbers.map((n) => n * n).toList();
  print(squared); // Output: [1, 4, 9, 16]
}
```

## Example — Combine Both

```
void main() {
  var fruits = ["apple", "banana", "mango"];

  // Using anonymous + arrow function
  fruits.forEach((fruit) => print(fruit.toUpperCase()));
}
```

**Output:**

```
APPLE
BANANA
MANGO
```

## Comments in Dart

Comments are notes in your code that help explain what it does.
They are **ignored by the Dart compiler**.

### Types of Comments

#### ➤ Single-line Comment

```
// This is a single-line comment
print("Hello World");
```

#### ➤ Multi-line Comment

```
/*
This is a multi-line comment.
It can cover multiple lines.
*/
print("Hello Dart");
```

#### ➤ Documentation Comment

Used for explaining functions, classes, or variables.
They start with `///` and are used for generating docs.

```
/// This function adds two numbers.
int add(int a, int b) {
  return a + b;
}
```

## Input / Output in Dart

### 1. stdin.readLineSync() — Taking Input in Dart

`stdin.readLineSync()` is used to **take input from the user** in the console.
It comes from the **dart:io** library.

**Syntax:**

```
String? variableName = stdin.readLineSync();
```

### Example:

```
import 'dart:io';

void main() {
  print("Enter your name:");
  String? name = stdin.readLineSync();  // takes input as string
  print("Hello, $name!");
}
```

### Notes:

- It **always returns a String? (nullable)** value.
- You can use `!` (null assertion) or check null before using it.

## 2. stdout.write() — Writing Output Without New Line

`print()` always moves to a **new line** after printing.
But `stdout.write()` prints output **without going to a new line**.
It is also from `dart:io`.

### Example:

```
import 'dart:io';

void main() {
  stdout.write("Enter your name: "); // stays on same line
  String? name = stdin.readLineSync();
  print("Hello, $name!");
}
```

Difference:

| Method | Moves to next line? | Example Output |
|---|---|---|
| print() | Yes | `Enter your name:`<br>`Neeraj` |
| stdout.write() | No | `Enter your name: Neeraj` |

### Notes:

- `int.parse()` converts String → Integer.
- The `!` means "I am sure it's not null".

# Basic CLI Input Handling

CLI = **Command Line Interface**
Dart supports reading input and writing output through the console (terminal).

Example with multiple inputs:

```
import 'dart:io';

void main() {
  print("Enter first number:");
  int num1 = int.parse(stdin.readLineSync()!);

  print("Enter second number:");
  int num2 = int.parse(stdin.readLineSync()!);

  print("Sum: ${num1 + num2}");
}
```

## Spread Operator ( . . . ) in Dart

The **spread operator** is used to **expand elements of a list, set, or map** into another list or map.
It's very useful when merging or combining collections.

#### Example 1 — Merging Lists

```
void main() {
  var list1 = [1, 2, 3];
  var list2 = [4, 5, 6];
  var combined = [...list1, ...list2];
  print(combined); // Output: [1, 2, 3, 4, 5, 6]
}
```

Here `...list1` means: "spread all items of list1 here."

#### Example 2 — Adding Items Dynamically

```
void main() {
  var baseList = [1, 2];
  var newList = [0, ...baseList, 3];
  print(newList); // [0, 1, 2, 3]
}
```

## Null-Aware Spread Operator (`...?`)

If you use a normal spread (. . .) on a **null list**, it causes an error.
To prevent that, use `...?` — it only spreads if the list is not null.

### Without Null-Aware Operator (Error)

```
void main() {
  List<int>? numbers;
  var list = [0, ...numbers]; // Error: numbers is null
}
```

### With Null-Aware Spread (Safe)

```
void main() {
  List<int>? numbers;
  var list = [0, ...?numbers];
  print(list); // Output: [0]
}
```

**Tip:**

- Use `...?` when your list or map might be null.
- It avoids runtime exceptions safely.

## Program: Check Even or Odd Number

Now let’s combine everything — input + output + logic.

**Example:**

```
import 'dart:io';

void main() {
  stdout.write("Enter a number: "); // input prompt
  int number = int.parse(stdin.readLineSync()!); // convert string to int

  if (number % 2 == 0) {
    print("$number is Even");
  } else {
    print("$number is Odd");
  }
}
```

**Sample Output:**

```
Enter a number: 7
7 is Odd
Enter a number: 12
12 is Even
```