1. Regular Functions

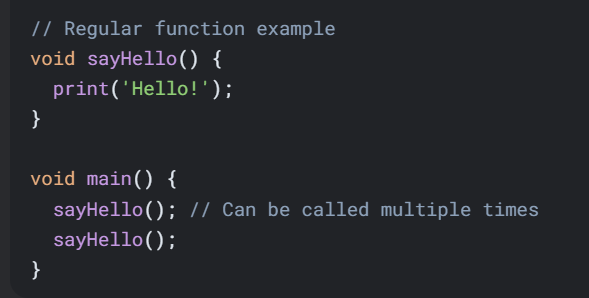
**Definition**: The most basic type of function that performs a specific task when called.

**Why we need it**:

* To organize code into reusable blocks
* To avoid code duplication
* To make code more readable and maintainable

**Where to use**:

* For any repetitive task in your program
* When you need to perform the same operation multiple times
* As building blocks for more complex programs



2. Functions with Parameters

**Definition**: Functions that accept input values (parameters) to customize their behavior.

**Why we need it**:

* To make functions more flexible and reusable
* To pass different data to the same function logic
* To create more dynamic behavior

**Where to use**:

* When you need similar operations on different data
* For calculations that require input values
* When processing user input or external data

A computer screen shot of a program code

AI-generated content may be incorrect.

3. Arrow Functions (=>)

**Definition**: A shorthand syntax for functions that contain a single expression.

**Why we need it**:

* For more concise code
* When the function body is very simple
* To improve readability for simple operations

**Where to use**:

* Short one-line functions
* Callbacks and simple transformations
* In functional programming patterns

A screenshot of a computer

AI-generated content may be incorrect.

4. Functions with Optional Parameters

**Definition**: Functions that can be called with some parameters omitted.

**Why we need it**:

* To provide default behavior when some values aren't specified
* To make function calls more flexible
* To maintain backward compatibility

**Where to use**:

* When some parameters have sensible defaults
* In APIs where not all parameters are always needed
* For configuration options



5. Named Parameters

**Definition**: Parameters that are specified by name when calling the function.

**Why we need it**:

* To make function calls more readable
* To allow parameters to be passed in any order
* For functions with many optional parameters

**Where to use**:

* When a function has multiple optional parameters
* For configuration objects
* In APIs where parameter order might be confusing

A computer screen with text

AI-generated content may be incorrect.

6. Return Values

**Definition**: Functions that compute and return a result to the caller.

**Why we need it**:

* To get results from computations
* To enable function composition
* To pass data between different parts of the program

**Where to use**:

* For calculations and data transformations
* When you need to use the result elsewhere
* In mathematical operations

A computer screen shot of a code

AI-generated content may be incorrect.

7. Implicit Return Type

**Definition**: When the return type is inferred by Dart from the function body.

**Why we need it**:

* For more concise code
* When the return type is obvious
* To reduce boilerplate

**Where to use**:

* In simple functions with obvious return types
* When prototyping
* In local helper functions

A screenshot of a computer

AI-generated content may be incorrect.

8. No Return Value (Void)

**Definition**: Functions that don't return any value.

**Why we need it**:

* For operations that perform actions rather than computations
* When the function's purpose is to produce side effects
* For procedures that don't need to communicate results

**Where to use**:

* For I/O operations
* When modifying object state
* For logging and printing

A screen shot of a computer program

AI-generated content may be incorrect.

9. Higher-Order Functions

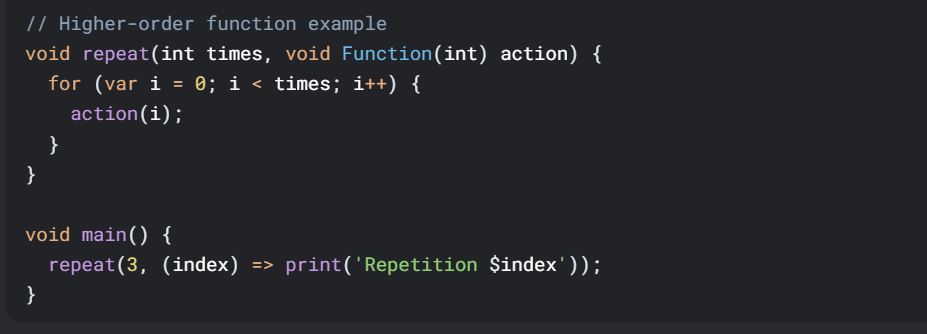
**Definition**: Functions that take other functions as parameters or return functions.

**Why we need it**:

* To enable functional programming patterns
* For creating abstractions over behavior
* To implement callbacks and event handlers

**Where to use**:

* In collection operations (map, filter, reduce)
* For asynchronous programming
* In strategy pattern implementations



10. Lexical Closures

**Definition**: Functions that capture variables from their surrounding scope.

**Why we need it**:

* To maintain state between function calls
* To create function factories
* For implementing callbacks with context

**Where to use**:

* In event handlers
* For creating specialized functions
* In stateful function patterns

A screen shot of a computer code

AI-generated content may be incorrect.

11. Generators

**Definition**: Functions that produce a sequence of values lazily.

**Why we need it**:

* For memory-efficient iteration
* To work with potentially infinite sequences
* For lazy evaluation

**Where to use**:

* When processing large datasets
* For mathematical sequences
* In stream-like data processing

A computer code on a black background

AI-generated content may be incorrect.