Higher-Order Functions with Lambda Expressions

What are Higher-Order Functions?

- **Higher-order functions** are functions that:
  - Take other functions as parameters
  - Return functions as results
  - Work with collections (Lists, Sets, etc.)

- In Dart, there are many higher-order functions.

```dart
// List has many higher-order functions
List<int> numbers = [1, 2, 3, 4, 5];
// Transform each item
numbers.map((x) => x * 2); // [2,4,6,8,10]
// Filter items
numbers.where((x) => x > 3); // [4,5]
// Folding (Combining) all items
numbers.fold(0, (sum, value) => sum + value);
// (0,1) => 1; (1, 2) => 3,
// (3, 3) => 6, (6, 4) => 10, (10, 5) => 15
```

The map() Function: Transform Everything

- **Purpose:** Transform each item in a list to something else

```dart
List<int> numbers = [1, 2, 3, 4, 5];
List<int> doubled = numbers.map((x) => x * 2).toList();
print(doubled); // [2, 4, 6, 8, 10]
```

```
var doubled = numbers.map((x) => x * 2).toList();
```

- In this example, a list of strings is transformed into a list of Text widgets.
- The expression items.map(...) returns an Iterable, not a List.
- We, we need to use toList().

Transforming to other types

```
List<String> prices = ['10', '20', '30'];
List<double> numbers = prices.map((price) =>
  double.parse(price)).toList();
// Result: [10.0, 20.0, 30.0]
```

- Converting Data Types: from a list of strings to a list of double-type values.

map() in Real Flutter Apps

```dart
List<String> fruits = ['Apple', 'Banana', 'Orange', 'Grape'];

Column(
  children: fruits.map((fruit) => Card(
    child: ListTile(
      title: Text(fruit),
      leading: Icon(Icons.food_bank),
    ),
  )).toList(),
)

List<String> items = ['Apple', 'Banana', 'Orange'];
// Creates: [Text('Hello Apple'),
//  Text('Hello Banana'),  Text('Hello Orange')]
items.map((item) => Text('Hello $item')).toList()
```

- **Creating a List of Cards and Texts**

The where() Function: Filter Your Data

- **Purpose:** Keep only items that match a condition

```dart
List<int> numbers = [1, 2, 3, 4, 5, 6];
List<int> evenNumbers = numbers.where((x) => x % 2 == 0).toList();
print(evenNumbers); // [2, 4, 6]

List<String> words = ['apple', 'banana', 'cat', 'dog'];
List<String> longWords = words.where((word) => word.length > 3).toList();
print(longWords); // ['apple', 'banana']
```

Filtering Search Results:

```
List<String> allFruits = ['Apple', 'Banana', 'Orange', 'Apricot'];
String searchQuery = 'Ap';

List<String> filteredFruits = allFruits
    .where((fruit) =>
      fruit.toLowerCase().
           contains(searchQuery.toLowerCase()))
    .toList();
// Result: ['Apple', 'Apricot']
```

- It finds only the strings that have "Ap" in them (case insensitive).

Combining map() and where()

```
List<int> numbers = [1, 2, 3, 4, 5, 6, 7, 8];

// Get even numbers, then double them
List<int> result = numbers
    .where((x) => x % 2 == 0)    // Filter: [2, 4, 6, 8]
    .map((x) => x * 2)           // Transform: [4, 8, 12, 16]
    .toList();
```

- **Filter then Transform:**

The fold() Function: Combine Everything

Purpose: Combine all items into a single value

```dart
List<int> numbers = [1, 2, 3, 4, 5];
int sum = numbers.fold(0,
  (total, current) => total + current);
print(sum);
// 0 -> 1 -> 3 -> 6 -> 10 -> 15
```

```dart
List<int> numbers = [1, 2, 3, 4, 5];
int sum = numbers.fold(0,
  (total, current) => total + current);

(0, 1) => (1, 2) => (3,3) => (6, 4) => (10, 5)
```

- The first value is the total sum so far.
- The second values are from the list.

```dart
// Concatenate strings
List<String> words = ['Hello', 'World', '!'];
String sentence = words.fold('',
  (result, word) => result + word + ' ');
print(sentence); // 'Hello World ! '
```

- We can use the fold to concatenate a list of strings into one string.

fold() for Complex Calculations

```dart
class Product {
  String name;
  double price;
  Product(this.name, this.price);
}

List<Product> cart = [
  Product('Apple', 1.5),
  Product('Banana', 2.0),
  Product('Orange', 3.0),
];

double totalPrice = cart.fold(0.0, (total, product) => total + product.price);
print('Total: \$${totalPrice}'); // Total: $6.5
```

- We can extract only the price of the Product to sum up all the product prices.

The reduce() Function: Simplified Combining

- **Purpose:** Like fold(), but uses first item as starting value

```dart
List<int> numbers = [1, 2, 3, 4, 5];
int sum = numbers.reduce((a, b) => a + b);
print(sum); // 15
// Find maximum
int max = numbers.reduce((a, b) => a > b ? a : b);
print(max); // 5
// Find minimum
int min = numbers.reduce((a, b) => a < b ? a : b);
print(min); // 1
```

The forEach() Function: Do Something with Each Item

Purpose: Execute code for each item (doesn't return anything)*

```
List<String> names = ['Alice', 'Bob', 'Charlie'];
// Print each name
names.forEach((name) => print('Hello, $name!'));
```

```dart
// More complex operations
List<Product> products = getProducts();
products.forEach((product) {
  print('Product: ${product.name}');
  print('Price: \$${product.price}');
  print('---');
});
```

forEach vs map

Quick Overview

Method	Purpose	Returns	Mutates Original
`forEach`	**Execute** side effects	`undefined`	Can mutate
`map`	**Transform** data	New array	Never mutates

Key Rule:

- Use `forEach` when you want to **DO** something
- Use `map` when you want to **GET** something

forEach - For Side Effects

```dart
// Wrong usage - trying to collect values
// Use map instead
List<int> numbers = [1, 2, 3, 4];
List<int> doubled = [];
numbers.forEach((num) {
  doubled.add(num * 2);  // Side effect
});
print(doubled); // [2, 4, 6, 8]

// Right usage - performing side effects
List<String> students = ['Alice', 'Bob', 'Charlie'];
students.forEach((student) {
  print('Welcome, $student!');     // Logging
  sendEmail(student);              // API call
  updateDatabase(student);         // Database update
});
```

Common Mistakes to Avoid

- Using a map without converting to a List.

```dart
List<int> numbers = [1, 2, 3];
var result = numbers.map((num) => num * 2);
print(result.runtimeType); // MappedListIterable<int, int>
// You get an Iterable, not a List!
```

- Using forEach to collect data.

```
List<int> numbers = [1, 2, 3];
List<int> doubled = [];
numbers.forEach((num) => doubled.add(num * 2)); // Awkward
```

Other Useful Higher-Order Functions

any() - Check if at least one item matches:

```dart
List<int> numbers = [1, 3, 5, 7, 8];
bool hasEven = numbers.any((x) => x % 2 == 0);
print(hasEven); // true (because of 8)
```

every() - Check if all items match:

```dart
List<int> ages = [18, 25, 30, 35];
bool allAdults = ages.every((age) => age >= 18);
print(allAdults); // true

List<String> emails = ['a@b.com', 'invalid-email', 'c@d.com'];
bool allValidEmails = emails.every((email) => email.contains('@'));
print(allValidEmails); // false
```

- The lambda expression should return true for all of the lists.

take() - Get first N items:

```
List<String> fruits = ['Apple', 'Banana', 'Orange', 'Grape', 'Mango'];
List<String> firstThree = fruits.take(3).toList();
print(firstThree); // ['Apple', 'Banana', 'Orange']
```

skip() - Skip first N items:

```
List<String> remaining = fruits.skip(2).toList();
print(remaining); // ['Orange', 'Grape', 'Mango']
```

takeWhile() - Take items while condition is true:

```
List<int> numbers = [1, 2, 3, 4, 5, 1, 2];
List<int> ascending = numbers.takeWhile((x) => x <= 3).toList();
print(ascending); // [1, 2, 3]
```