1. Add name to a `List`:

```
void main() {
  List<String> names = ['Alice', 'Bob'];
  names.add('Charlie'); // Add new name
  print(names); // Output: [Alice, Bob, Charlie]
}
```

✅ 2. Add name into a `Map`:

Maps use key-value pairs, like `"id": "name"` or `"name1": "Alice"`.

```
void main() {
  Map<int, String> nameMap = {1: 'Alice', 2: 'Bob'};
  nameMap[3] = 'Charlie'; // Add a new key-value pair
  print(nameMap); // Output: {1: Alice, 2: Bob, 3: Charlie}
}
```

—--

Directly using string index

```
void main() {
  String name = 'Ansila';

  print(name[0]);
  print(name[1]);
  print(name[2]);
  print(name[3]);
  print(name[4]);
  print(name[5]);
}
```

🔸 **Output:**

```
A
n
s
i
l
A
```

List of Maps

➤ A list that contains multiple maps (commonly used for records like users, products, etc.)

📌 **Example:**

```
void main() {
  List<Map<String, dynamic>> students = [
    {'name': 'Ameer', 'age': 20},
    {'name': 'Niya', 'age': 22},
    {'name': 'Ravi', 'age': 21},
  ];

  print(students[0]['name']); // Output: Ameer
  print(students[1]['age']);  // Output: 22
}
```

Nested Lists

➢ A list inside another list.

📌 **Example:**

```
void main() {
  List<List<int>> matrix = [
    [1, 2],
    [3, 4],
    [5, 6],
  ];

  print(matrix[0]);       // Output: [1, 2]
  print(matrix[1][1]);    // Output: 4
}
```

- matrix[1][1] → 2nd row, 2nd column (value = 4)

➢ **add()**, **addAll()**, **insert()**, and **other common List methods** in Dart — all related to adding numbers.

✅ 1. add() → Add one item

```
void main() {
  List<int> numbers = [];

  numbers.add(10);
  numbers.add(20);

  print(numbers);
}

// Output: [10, 20]
```

✅ 2. `addAll()` → Add multiple items at once

```
void main() {
  List<int> numbers = [1, 2];

  numbers.addAll([3, 4, 5]);

  print(numbers);
}

// Output: [1, 2, 3, 4, 5]
```

✅ 3. `insert(index, value)` → Add item at a specific position

```
void main() {
  List<int> numbers = [1, 3, 4];

  numbers.insert(1, 2); // Insert 2 at index 1

  print(numbers); // Output: [1, 2, 3, 4]
}
```

✅ 4. `insertAll(index, [values])` → Add multiple items at a specific position

```
void main() {
  List<int> numbers = [1, 5];

  numbers.insertAll(1, [2, 3, 4]); // Insert at index 1
```

```
  print(numbers); // Output: [1, 2, 3, 4, 5]
}
```

✅ 5. `setAll(index, [values])` → Replace elements starting at a specific index

```
void main() {
  List<int> numbers = [10, 20, 30, 40];

  numbers.setAll(1, [21, 31]); // Replace from index 1

  print(numbers); // Output: [10, 21, 31, 40]
}
```

🧠 Summary of List Add Methods

Method	Use
`add(x)`	Add a single item
`addAll([x,y])`	Add multiple items
`insert(i, x)`	Add item at specific index
`insertAll(i, [x,y])`	Add multiple at specific index
`setAll(i, [x,y])`	Replace items starting at index

✅ 6. `remove(value)`

👉 Removes the first matching value from the list.

```
void main() {

  List<int> numbers = [10, 20, 30, 40];

  numbers.remove(30); // Removes the value 30

  print(numbers); // Output: [10, 20, 40]

}
```

Explanation:
This removes the value `30` from the list. If there are duplicates, only the first one will be removed.

✅ 7. `removeAt(index)`

👉 Removes the value at a specific index.

```
void main() {

List<int> numbers = [10, 20, 30];

numbers.removeAt(1); // Removes item at index 1 (20)

print(numbers); // Output: [10, 30]

}
```

Explanation:
Removes the item at index 1, which is `20`.

✅ 8. `removeLast()`

👉 Removes the last item in the list.

```
void main() {

  List<int> numbers = [10, 20, 30];

  numbers.removeLast();

  print(numbers); // Output: [10, 20]

}
```

✅ 9. Update a value using index

👉 Change a value in the list by directly accessing the index.

```
void main() {

  List<String> names = ['Alice', 'Bob', 'Charlie'];

  names[1] = 'Benny'; // Replace 'Bob' with 'Benny'

  print(names);

}

 // Output: [Alice, Benny, Charlie]
```

Explanation:
We updated the value at index 1 to `'Benny'`.

✅ 10. `sort()`

👉 Sorts the list in ascending order.

```
void main() {

  List<int> numbers = [30, 10, 50, 20];

  numbers.sort();

  print(numbers);

// Output: [10, 20, 30, 50]}
```

🔁 Bonus: `reversed`

👉 Reverses the list (returns an iterable, so use `toList()`).

```
void main() {

  List<int> numbers = [1, 2, 3];

  print(numbers.reversed.toList());

}

// Output: [3, 2, 1]
```

🧠 Summary Table

Method	Description
`remove(value)`	Removes the first occurrence of a value
`removeAt(index)`	Removes item at a specific index
`removeLast()`	Removes the last item in the list
`list[index] = x`	Updates an item at a given index
`sort()`	Sorts the list in ascending order
`reversed`	Reverses the list (use `.toList()` to print)

➢ Searching / Filtering List

📌 **Example 1: Filter students above age 21.**

```
void main() {

  List<Map<String, dynamic>> students = [

    {'name': 'Ameer', 'age': 20},

    {'name': 'Niya', 'age': 22},

    {'name': 'Ravi', 'age': 23},

  ];

var filtered = students.where((student) => student['age'] > 21).toList();

print(filtered);

}

 // Output: [{'name': 'Niya', 'age': 22}, {'name': 'Ravi', 'age': 23}]
```

🔍 Explanation:

- `.where()` is used to apply a condition and filter the list.
- `.toList()` converts the filtered result into a proper List.

In this example, it filters students whose age is greater than 21.

📌 **Example 2: Search for names that contain the letter `'a'`.**

```
void main() {

List<String> names = ['Ameer', 'Niya', 'Ravi'];

var result = names.where((name) => name.contains('a')).toList();

print(result); // Output: [Niya, Ravi]

}
```

🔍 Explanation:

- `name.contains('a')` checks if the name has the letter 'a'.
- Only those names are selected and returned as a new list.

🧠 Summary Table

Feature	What it does
List of Maps	Stores structured data (like records/objects)
Nested Lists	Stores lists inside lists (like tables/matrices)
`.where()`	Filters list based on a condition
`.contains()`	Checks if a value or letter exists inside a string

What is `split()` in Dart?

- `split()` is a **String method** used to **break a string into a list**, based on a given **separator** (like space, comma, etc.).

✅ 1. Basic Example – Split by space `' '`

```
void main() {
  String sentence = 'I love Flutter';

  List<String> words = sentence.split(' ');

  print(words);
}
// Output: [I, love, Flutter]
```

✅ 2. Split by comma `','`

```
void main() {
  String data = 'apple,banana,kiwi';

  List<String> fruits = data.split(',');

  print(fruits);
}
 // Output: [apple, banana, kiwi]
```

✅ 3. Split into individual characters using `''` (empty string)

```
void main() {
  String word = 'Hello';

  List<String> letters = word.split('');

  print(letters);
}
// Output: [H, e, l, l, o]
```

✅ 4. Split by a special character (like - or |)

```
void main() {
  String code = '123-456-789';

  List<String> parts = code.split('-');

  print(parts);
}
// Output: [123, 456, 789]
```

🧠 Summary Table:

String	Code	Output
'a b c'	split(' ')	[a, b, c]
'1,2,3'	split(',')	[1, 2, 3]
'Hello'	split('')	[H, e, l, l, o]
'x-y-z'	split('-')	[x, y, z]

Common Dart List Utility Methods

`contains()`, `indexOf()`, `isEmpty()`, `clear()`, `length()`, `sublist()` – all these are considered Common Dart List Utility Methods.

These headings cover all the methods like:

- Checking if value exists (`contains`)
- Finding index (`indexOf`, `lastIndexOf`)
- Checking empty (`isEmpty`)
- Removing all (`clear`)
- Getting count (`length`)
- Slicing part of list (`sublist`)

✅ 1. `contains(value)`

👉 Returns `true` if the value exists in the list.

```
void main() {
  List<String> fruits = ['apple', 'banana', 'kiwi'];

  print(fruits.contains('banana')); // true
  print(fruits.contains('orange')); // false
}
```

Explanation:
contains() checks whether a value exists in the list.
It returns **true** if the value is found, otherwise **false**.

✅ 2. indexOf(value)

👉 Returns the **index** where the value is found.

```
void main() {
  List<int> numbers = [10, 20, 30, 40];

  print(numbers.indexOf(30)); // Output: 2
}
```

Explanation:
The value 30 is at **index 2**.
indexOf() helps you find **where** a value is located.

✅ 3. lastIndexOf(value)

👉 Returns the **last index** where the value is found (if duplicates exist).

```
void main() {
  List<int> nums = [5, 10, 5, 20];
```

```
  print(nums.lastIndexOf(5)); // Output: 2
}
```

✅ 4. isEmpty / isNotEmpty

```
void main() {
  List<int> a = [];
  List<int> b = [1, 2];

  print(a.isEmpty);     // true
  print(b.isNotEmpty);  // true
}
```

Explanation:

- `isEmpty` → checks whether the list is empty
- `isNotEmpty` → returns `true` if the list has at least one item

✅ 5. clear()

👉 Removes all items and makes the list empty

```
void main() {
  List<int> nums = [1, 2, 3];

  nums.clear();

  print(nums); // Output: []
}
```

✅ 6. length

👉 Returns the **number of elements** in the list

```
void main() {
  List<String> colors = ['red', 'green', 'blue'];

  print(colors.length); // Output: 3
}
```

✅ 7. sublist(start, end)

👉 Returns a **slice** (portion) of the list from start index to end-1

```
void main() {
  List<int> nums = [10, 20, 30, 40, 50];

  print(nums.sublist(1, 4)); // Output: [20, 30, 40]
}
```

Explanation:
It extracts a part of the list from index **1 to 3** (index 4 is not included).

🔚 Summary Table

Method	Purpose / Use
`contains(x)`	Check if `x` exists in the list
`indexOf(x)`	Find the index of the first occurrence of `x`
`lastIndexOf(x)`	Find the last index of `x` in the list
`isEmpty` / `isNotEmpty`	Check if list is empty / has elements
`clear()`	Delete all elements from the list
`length`	Get the number of elements in the list
`sublist(a, b)`	Get a portion of the list from `a` to `b-1`