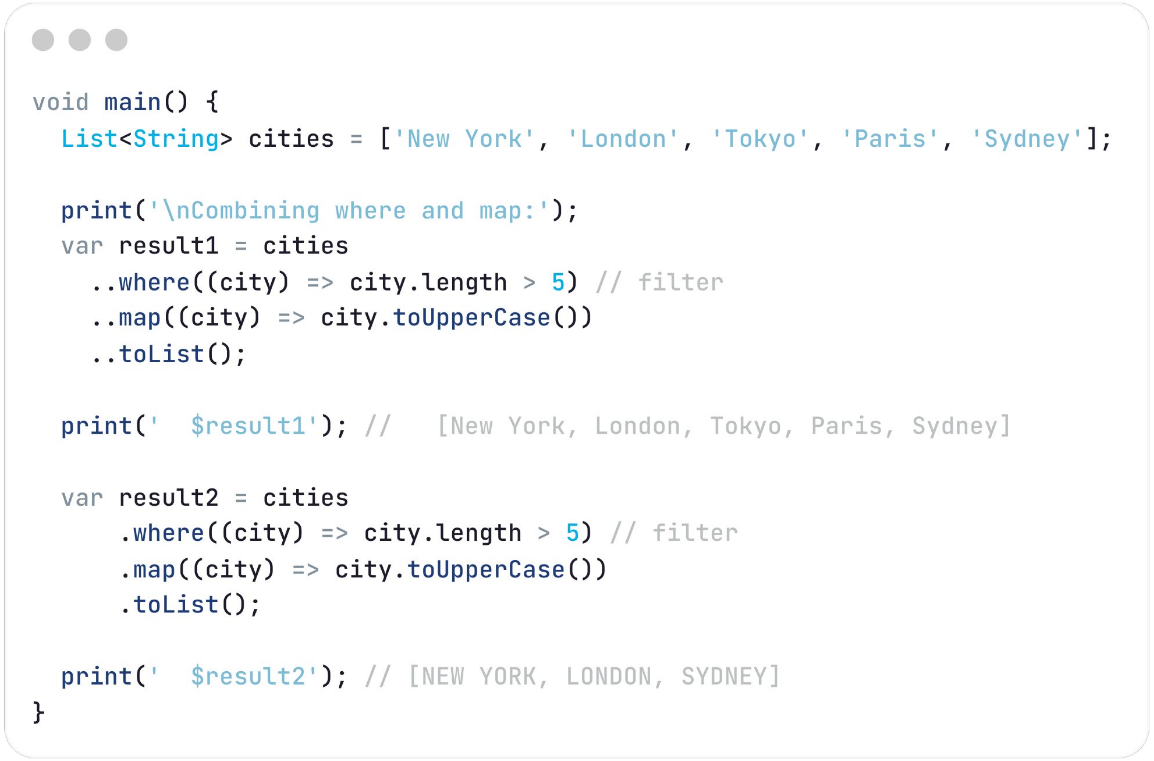
Regular Dot vs Cascade Operator

Dot (.) = "Do this, then work with the result"
Cascade (..) = "Do this to the same object, then do more"

Problem Statement



```dart
void main() {
  List<String> cities = ['New York', 'London', 'Tokyo', 'Paris', 'Sydney'];

  print('\nCombining where and map:');
  var result1 = cities
    ..where((city) => city.length > 5) // filter
    ..map((city) => city.toUpperCase())
    ..toList();

  print('  $result1'); //   [New York, London, Tokyo, Paris, Sydney]

  var result2 = cities
      .where((city) => city.length > 5) // filter
      .map((city) => city.toUpperCase())
      .toList();

  print('  $result2'); // [NEW YORK, LONDON, SYDNEY]
}
```

Regular Dot Operator (.) - 1/2

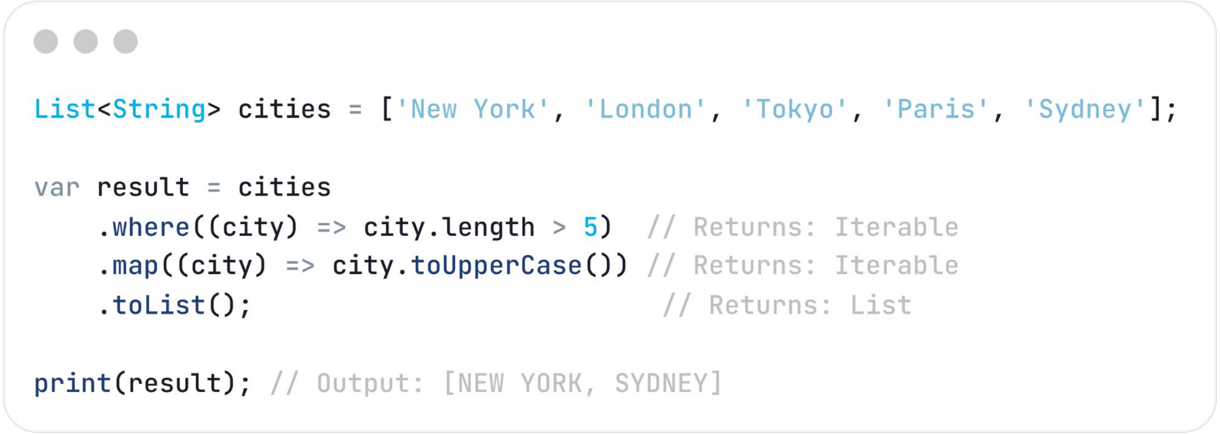
What it does:

- **Returns the result** of each method call
- Enables **method chaining** on different objects
- Each method operates on the **return value** of the previous method

Regular Dot Operator (.) - 2/2

Flow:

- `cities.where()` → Returns filtered Iterable
- `.map()` operates on the Iterable (not cities)
- `.toList()` operates on the mapped Iterable



```dart
List<String> cities = ['New York', 'London', 'Tokyo', 'Paris', 'Sydney'];

var result = cities
    .where((city) => city.length > 5)  // Returns: Iterable
    .map((city) => city.toUpperCase()) // Returns: Iterable
    .toList();                          // Returns: List

print(result); // Output: [NEW YORK, SYDNEY]
```

Cascade Operator (..) - 1/2

What it does:

- **Returns the original object**, not the method result
- Allows multiple operations on the **same object**
- Useful for object configuration/initialization

Cascade Operator (..) - 2/2

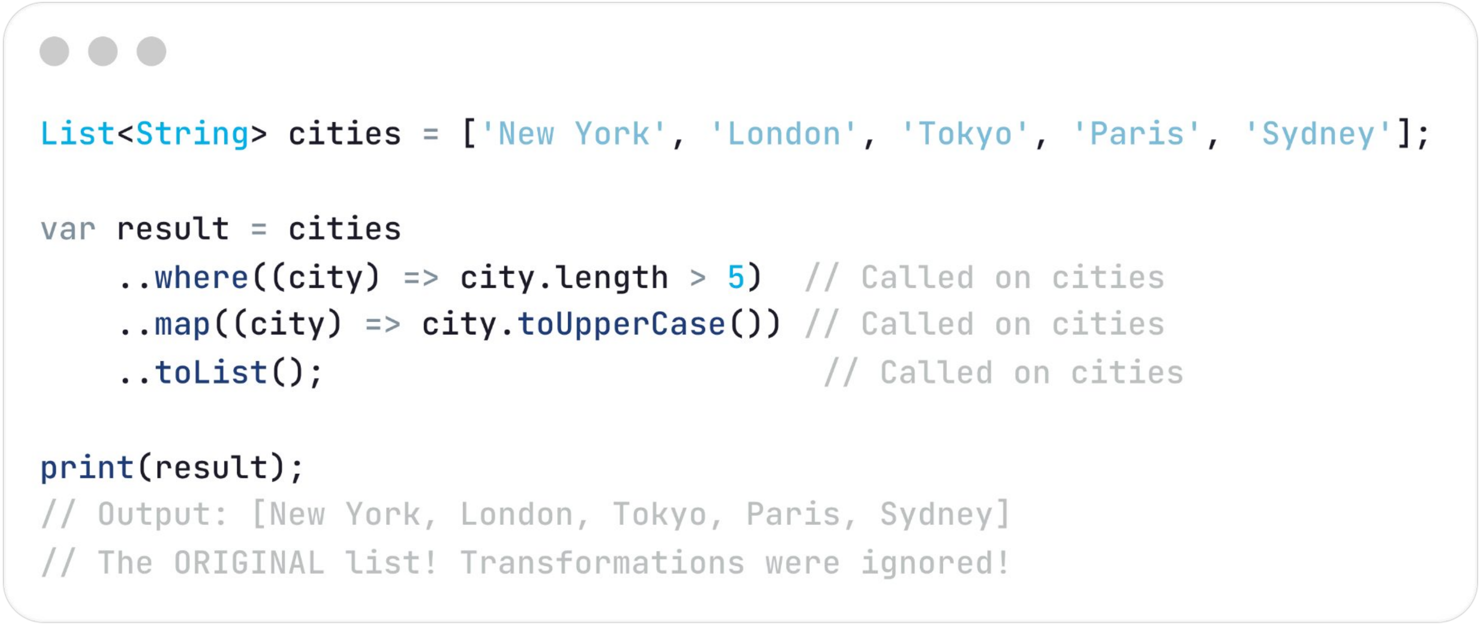
```dart
class Button {
  String? text;
  String? color;
  Function? onClick;
}

// Using cascade operator
var button = Button()
  ..text = 'Click me'
  ..color = 'Blue'
  ..onClick = () => print('Clicked!');

// Equivalent to:
var button = Button();
button.text = 'Click me';
button.color = 'Blue';
button.onClick = () => print('Clicked!');
```

What Happens with Wrong Usage? - 1/2

Using Cascade Where Regular Dot is Needed:



```dart
List<String> cities = ['New York', 'London', 'Tokyo', 'Paris', 'Sydney'];

var result = cities
    ..where((city) => city.length > 5)  // Called on cities
    ..map((city) => city.toUpperCase()) // Called on cities
    ..toList();                          // Called on cities

print(result);
// Output: [New York, London, Tokyo, Paris, Sydney]
// The ORIGINAL list! Transformations were ignored!
```

What Happens with Wrong Usage? - 2/2

Why it fails:

1. **`cities..where()`** → executes but **returns cities**
2. **`cities..map()`** → executes but **returns cities**
3. **`cities..toList()`** → tries to call on List (may error)
4. result = original cities list (unchanged!)

Quick Reference & Best Practices

Use Regular Dot (.) when:

- Chaining transformations on different return values
- Working with streams, iterables, futures
- Each step produces a new object to work with

Example: `list.where().map().toList()`

Use Cascade Operator (..) when:

- Setting multiple properties on the same object
- Calling multiple methods on the same object
- Building/configuring objects

Example: `object..prop1 = x..prop2 = y..method()`