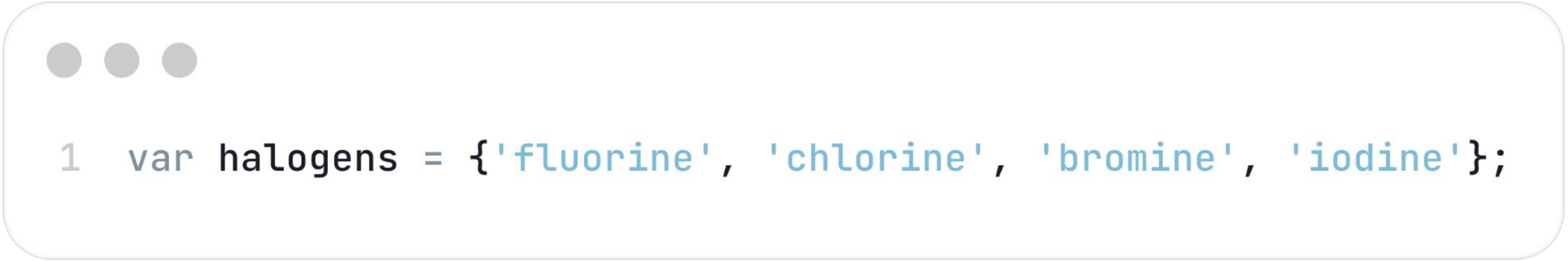
Sets: Unique Elements

Ensuring uniqueness of elements
Topics: union, intersection, difference

What is set?

In computer science, a set is defined as a data structure that stores a collection of distinct elements.

Dart has built in support for set. A set in Dart is an unordered collection of unique elements.



```dart
var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine'};
```

Types of Set

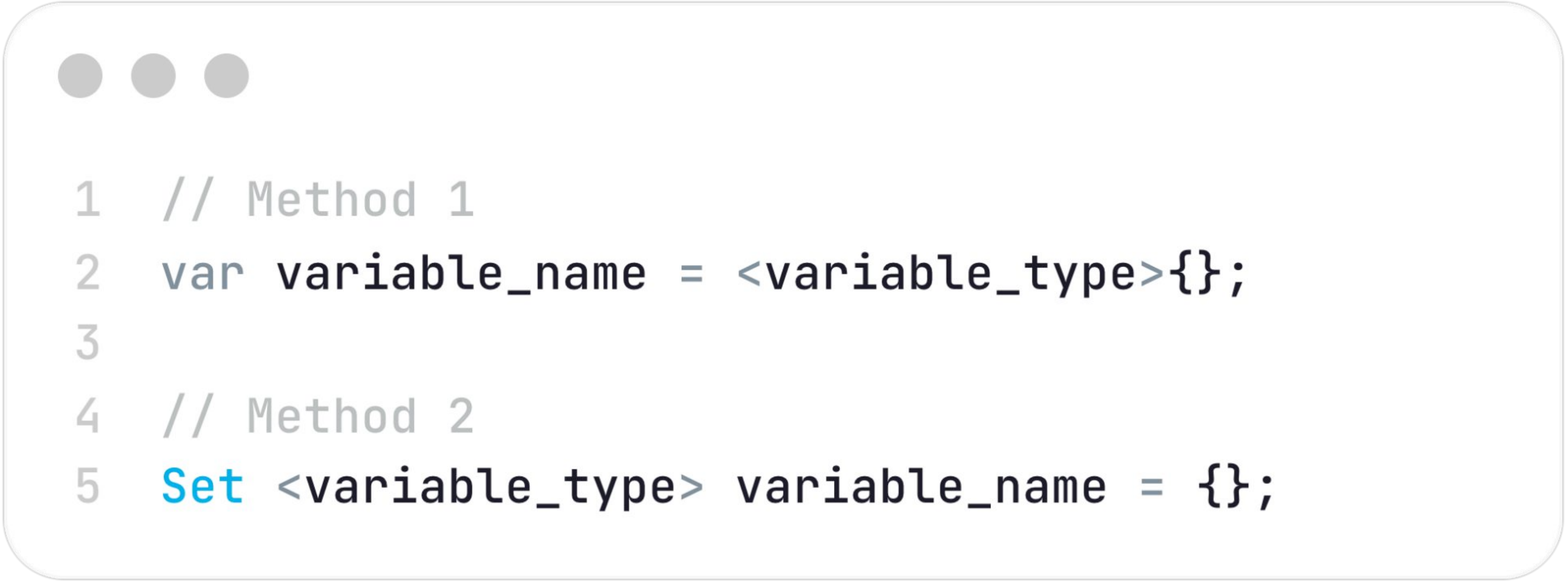
There are two types of sets:

1. Unordered Set
2. Ordered Set

Dart doesn't support ordered set.

Declaring Set

There are two ways to declare set:



```
// Method 1
var variable_name = <variable_type>{};

// Method 2
Set <variable_type> variable_name = {};
```

Set Declaration Styles

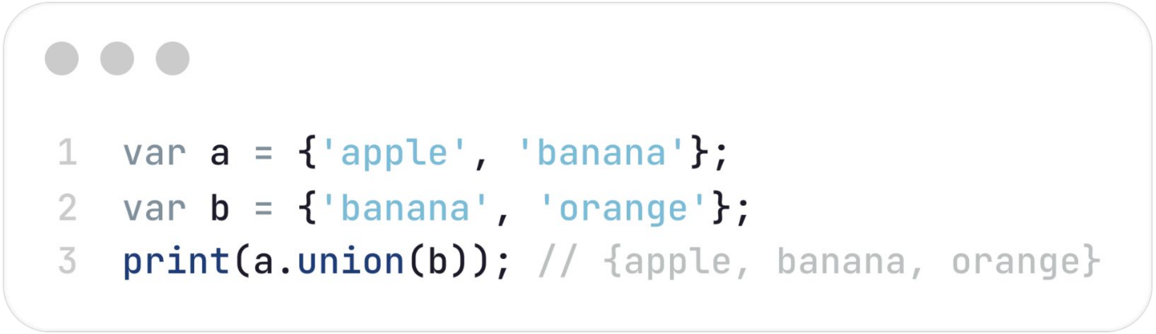
```dart
// Method 1: Using 'var'
var fruits = <String>{};        // Empty Set of Strings
fruits.add('Apple');
fruits.add('Banana');
fruits.add('Apple');            // Duplicate ignored

print(fruits); // {Apple, Banana}

// Method 2: Using 'Set<Type>'
Set<String> usernames = {};     // Empty Set of Strings
usernames.add('alice');
usernames.add('bob');
usernames.add('alice');         // Duplicate ignored

print(usernames); // {alice, bob}
```

Set Operations in Dart

- Dart Set supports common mathematical operations:
    - **union()** → combine sets
    - **intersection()** → common elements
    - **difference()** → elements in one but not the other
- Useful for filtering, merging, or comparing data.

Set Operation: Union

Definition: Combines all unique elements from both sets.



```
var a = {'apple', 'banana'};
var b = {'banana', 'orange'};
print(a.union(b)); // {apple, banana, orange}
```

Merges both sets, removing duplicates.

Set Operation: Intersection

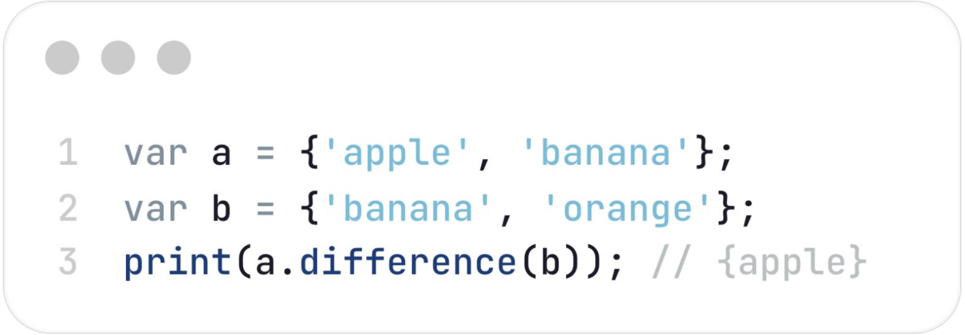
Definition: Returns elements common to both sets.

```
var a = {'apple', 'banana'};
var b = {'banana', 'orange'};
print(a.intersection(b)); // {banana}
```

Keeps only matching elements.

Set Operation: Difference

Definition: Returns elements in one set but not in the other.



```
var a = {'apple', 'banana'};
var b = {'banana', 'orange'};
print(a.difference(b)); // {apple}
```

Filters out elements present in the second set.

Real-World Use Cases of Sets

- Useful for working with unique data
- Helps in filtering, comparison, and analytics
- Common in apps, websites, and backend systems

Real-World Use Cases of Union – Merging Data

- Combine users from multiple platforms (e.g., app + web)a
- Merge tags or categories from different sources
- Gather unique IDs from multiple lists

Real-World Use Cases of Intersection – Finding Common Items

- Find mutual friends between users
- Identify common interests or skills
- Detect shared access between roles or teams

Real-World Use Cases of Difference – Filtering Out Data

- Find users exclusive to one platform
- Remove already processed items from a new batch
- Identify missing permissions or roles

Closing

- Set operations make data handling simpler and faster
- Ideal for unique data, comparisons, and analytics
- Think of:
  - Union → combine
  - Intersection → common
  - Difference → exclude