I'm human

Dart list methods

Dart Fundamentals: Getters and Setters, Essential List/Array Methods ---------- As I continue to explore Flutter, I've been focusing on grasin' the Dart language basics. One area that's been puzzlin' me is array methods. As a web developer with a JS background, I found that Dart's list/array methods are quite similar. In this post, I'll cover the essentials for my own reference and anyone else who might be in the same boat. **Prerequisites** To follow along, you should have some basic knowledge of Flutter/Dart OOP. You can use DartPad, a playground for Dart, to experiment with the code examples. **Getting Started** Let's start with a simple example: ```dart void main() { List fruits = ['mangoes', 'bananas', 'pears', 'oranges']; print(fruits); } ``` Notice how we've used type annotation for the `fruits` variable? This improves readability and helps with IntelliSense in code editors like Visual Studio Code. **Type Safety** The above code is type-safe, meaning you can only assign strings to the `fruits` variable. If you try to add items of different types, it will still work, but using `List` would allow that. **const and final** Let's explore what happens when we mark our list as either `final` or `const`. Try changing the code to: ```dart void main() { final List fruits = ['mangoes', 'bananas', 'pears', 'oranges']; fruits.add('apples'); print(fruits); } ``` Run it and see what happens. Marking as `final` means we can't reassign the variable, but we can still mutate the list. Now, let's try something else: ```dart void main() { final List fruits = ['mangoes', 'bananas', 'pears', 'oranges']; fruits = [...fruits, 'apples']; print(fruits); } ``` This will give you an error. To make the list immutable, use the `const` keyword: ```dart void main() { final List fruits = const ['mangoes', 'bananas', 'pears', 'oranges']; //fruits.add('apples'); print(fruits); } ``` There are two ways to implement this. I'll cover the differences in more detail later. **Creating Lists** We've already seen one way of creating lists using the `List()` constructor. Let's explore a few more methods: ```dart void main() { var list = List(5); print(list); //output - [null, null, null, null, null] } ``` This creates a list of length 5 with null as its items/values. Next up: essential Dart list/array methods! It seems that you are trying to learn about Dart's built-in list operations. Let's take a look at some of the most commonly used methods and properties. Firstly, `List.filled()` is used to create a list filled with a specified value up to a certain length. You can also specify whether the list should be growable or not. For example: `var list = List.filled(5, "hello", growable: true);`. Another way to create a list is using `List.generate()`, which takes in a length and a generator function that returns the value for each index. This method also accepts an optional parameter for growability. `List.from()` creates a new list from another iterable, and it also has an optional parameter for specifying whether the resulting list should be growable or not. Now let's talk about some of the most commonly used methods on lists: - `add()`: adds an item to the end of a growable list. - `remove()`: removes the first occurrence of an object from a growable list. If there are duplicate values, it will only remove one instance. - `removeAt()`: removes the item at a specified index in a growable list. - `addAll()`: adds all items from another iterable to the end of a growable list. Some other methods that are worth mentioning include: - `asMap()`: returns a map representation of the list, where indices are keys and values are items. - `insert()` and `insertAll()`: similar to `add()` and `addAll()`, but they allow you to specify an index at which to insert the item(s). - `getRange()`: returns an iterable containing items within a specified range. You can convert this to a list using `toList()`. Lastly, there's `fillRange()`, but its documentation is incomplete in your snippet. Given article text here void main() { var numbers = [30,10,22,45,24,88,1,37,100,0]; print(numbers.reduce((a,b)=> a + b)); } Dart programming language features a list class that provides various methods for manipulating and operating on lists. Some essential methods include map, which applies a given function to every item in the list, followed by a call to toList() to convert it back into a list. Another method is forEach, which takes each item in the list and applies the given function in their index order. Additionally, there are methods like followedBy, take, and others that can be used to manipulate lists. The language also supports getters and setters, as well as different types of lists such as fixed-length and growable lists. and where, the element is in the list, use indexOf or lastIndexOf. final indexA = growableList.indexOf('A'); // -1 (not in the list) final firstIndexB = growableList.indexOf('B'); // 1 final lastIndexB = growableList.lastIndexOf('B'); // 4 To remove an element from the growable list, use remove, removeAt, removeLast, removeRange or removeWhere. growableList.remove('C'); growableList.removeLast(); print(growableList); // [G, B, X] To insert an element at position in the list, use insert or insertAll. growableList.insert(1, 'New'); print(growableList); // [G, New, B, X] To replace a range of elements in the list, use fillRange, replaceRange or setRange. growableList.replaceRange(0, 2, ['AB', 'A']); print(growableList); // [AB, A, B, X] growableList.fillRange(2, 4, 'F'); print(growableList); // [AB, A, F, F] To sort the elements of the list, use sort. growableList.sort((a, b) => a.compareTo(b)); print(growableList); // [A, AB, F, F] To shuffle the elements of this list randomly, use shuffle. growableList.shuffle(); print(growableList); // e.g. [AB, F, A, F] To find the first element satisfying some predicate, or give a default value if none do, use firstWhere. bool isVowel(String char) => char.length == 1 && "AEIOU".contains(char); final firstVowel = growableList.firstWhere(isVowel, orElse: () => ''); // '' There are similar lastWhere and singleWhere methods. A list is an Iterable and supports all its methods, including where, map, whereType and toList. Lists are Iterable. Iteration occurs over values in index order. Changing the values does not affect iteration, but changing the valid indices—that is, changing the list's length—between iteration steps causes a ConcurrentModificationError. This means that only growable lists can throw ConcurrentModificationError. If the length changes temporarily and is restored before continuing the iteration, the iterator might not detect it. It is generally not allowed to modify the list's length (adding or removing elements) while an operation on the list is being performed, for example during a call to forEach or sort. Changing the list's length while it is being iterated, either by iterating it directly or through iterating an Iterable that is backed by the list, will break the iteration. Lists in Dart are a powerful data structure that can store multiple values. They are zero-based, meaning the first element is at index 0, and you can access or modify elements using subscript notation (`listName[index]`). Lists have methods to add or remove elements. To create a list, you simply enclose items within square brackets: `var scores = [1, 3, 4, 2];`. The type of the elements in this example is inferred as integers because all the initial values are integers. Dart will also infer the type if you initialize it with an empty list like `var scores = [];` but this would make it dynamic, losing some benefits of type safety. You can print a list to see its content using the `print()` function: `void main() { var scores = [1, 3, 4, 2]; print(scores); }`. The output will be `[1, 3, 4, 2]`. Elements are accessed and modified using their index within square brackets. For instance, to access the third element of a list named `scores`, you would use `scores[2]`. To add an item at the end of a list, you can use the `add()` method: `scores.add(5);`. This operation does not change the existing elements' positions but adds a new one at the end. If you need to remove an element from a list, you can use the `remove()` method. For example, removing number 1 from the scores list is done by `scores.remove(1);`. For lists that should not be reassigned (like if they're part of another data structure), use the `final` keyword: `final scores = [1, 3, 4, 2];`. This means once you've set its value, it cannot be changed. If your intention is to make a list truly immutable, use the `const` keyword instead of `final`: `void main() { const scores = [1, 3, 4, 2]; scores.add(6); }`. However, using `const` will prevent modifying the list even with methods like `add()` or `remove()`, resulting in an error. To access or manipulate elements within a Dart list, several properties and methods are available. The length property is used to determine the number of elements in a list, as demonstrated by `var scores = [1, 3, 4, 2, 5]; print('Length: ${scores.length}');`. To access specific positions or elements within a list, you can use the first and last properties. For example, to get the first and last elements of a list, you would use `print('First: ${scores.first}');` and `print('Last: ${scores.last}');`, respectively. Checking if a list is empty or contains any elements can be done using the isEmpty and isNotEmpty properties. If a list is empty, it will return true for isEmpty and false for isNotEmpty. For iteration over list elements, both for and for-in loops are available in Dart. The traditional for loop involves keeping track of an index variable to access elements at specific positions within the list. A more concise approach is using the for-in loop that directly assigns each element from the list to a variable during each iteration. The forEach method of List also allows executing a function for each element without needing to manually keep track of indices or use direct assignment in loops, as shown by `scores.forEach((score) => print(score));`. Additionally, Dart supports combining multiple lists into one using the spread operator (...). This can be seen in examples where lower and upper lists are combined into scores list. Lastly, the collection if can be used to conditionally include elements within a list. It allows for dynamic construction of lists based on certain conditions, as demonstrated by `var greetings = [ if (bye) 'Good Bye', 'Hi', 'Hi there', ];`. This feature enables more flexible and dynamic list creation in Dart. void main() { var numbers = [1, 2, 3]; var scores = [0, for (var number in numbers) number * 2]; print(scores); } List is one of four types of collection Dart offers. It is equivalent to Array and is an ordered collection of items, starting with index 0. In this article, we'll take a look at various List methods that may not be used frequently but are very useful in retrieving data for unique cases, with example of each. sublist(): This method returns a new list containing elements from index between start and end. Note that end element is exclusive while start is inclusive. var myList= [1,2,3,4,5]; print(myList.sublist(1,3)); // [2,3] If end parameter is not provided, it returns all elements starting from start till length of the list.print(myList.sublist(1)); // [2,3,4,5] shuffle(): This method re-arranges order of the elements in the given list randomly. myList.shuffle(); print('$myList'); // [5,4,3,1,2] reversed: reversed is a getter which reverses iteration of the list depending upon given list order. var descList= [6,5,4,3,2,1]; print(descList.reversed.toList()); // [1,2,3,4,5,6] var ascList = [1,2,3,4,5,6]; print(ascList.reversed.toList()); // [6,5,4,3,2,1] asMap(): This method returns a map representation of the given list. The key would be the indices and value would be the corresponding elements of the list. List sports = ['cricket', 'football', 'tennis', 'baseball']; Map map = sports.asMap(); print(map); // {0: cricket, 1: football, 2: tennis, 3: baseball} whereType(): This method returns iterable with all elements of specific data type. Let's say we have a list with mix data such as String and int and we just want to retrieve int data from it, then we would use whereType method as: var mixList = [1, "a", 2, "b", 3, "c", 4, "d"]; var num = mixList.whereType(); print(num); // (1, 2, 3, 4) getRange(): This method returns elements from specified range [start] to [end] in same order as in the given list. Note that, start element is inclusive but end element is exclusive. var myList = [1, 2, 3, 4, 5]; print(myList.getRange(1,4)); // (2, 3, 4) replaceRange(): This method helps to replace / update some elements of the given list with the new ones. The start and end range need to be provided alongwith the value to be updated in that range. var rList= Given article text here List methods explained var rList = [0,1,10,3,4,5]; rList.replaceRange(2,3, [10]); print('$rList'); // [0, 1, 10, 3, 4, 5] firstWhere():This method returns the first element from the list when the given condition is satisfied. var firstList = [1,2,3,4,5,6]; print(firstList.firstWhere((i) => i < 4)); // 1 var sList = ['one', 'two', 'three', 'four']; print(sList.firstWhere((i) => i.length > 3)); // three lastWhere() returns last element from the list when given condition is met. where() returns new list that matches the condition. singleWhere():This method returns the first matching element from the list when there's an exact match. var mList = [1,2,3,4]; print(mList.singleWhere((i) => i == 3)); // 3 If the given list contains a duplicate, then singleWhere method retuns an exception. In that case, we can use firstWhere method which returns the first matching element irrespective of repeating / duplicates in the list. var sList = [1, 2, 3, 3, 4]; print(sList.singleWhere((i) => i == 3)); // Bad state: Too many elements print(sList.firstWhere((i) => i == 3)); // 3 fold():This method returns a single value by iterating all elements of given list along with an initialValue , ie, if we want sum of all elements or product of all elements, then, fold helps us to do that. var lst = [1,2,3,4,5]; var res = lst.fold(5, (i, j) => i + j); print('res is ${res}'); // res is 20 reduce():This method is very similar to fold and returns a single value by iterating all elements of given list. Only difference is, this method doesn't take any initialValue and the given list should be non-empty. var lst = [1,2,3,4,5]; var res = lst.reduce((i, j) => i + j); print('res is ${res}'); // res is 15 followedBy():This method appends new iterables to the given list. var sportsList = ['cricket', 'tennis', 'football']; print(sportsList.followedBy(['golf', 'chess']).toList()); // [cricket, tennis, football, golf, chess] any():This method returns a boolean depending upon whether any element satisfies the condition or not. print(sportsList.any((e) => e.contains('cricket'))); // true every():This method returns a boolean depending upon whether all elements satisfies the condition or not. print(sportsList.every((e) => e.startsWith('a'))); // false take():This method returns iterable starting from index 0 till the count provided from given list. print(sportsList.take(2)); // (cricket, tennis) skip():This method ignores the elements starting from index 0 till count and returns remaining iterable from given list. print(sportsList.skip(2)); // (football) last element from a given list gets cleared by the 'clear' method. or list.clear() removes all items from the list.

List properties and methods in dart. List all methods in dart. Dart where method. Dart list example.