M30299– Programming

moodle.port.ac.uk

Week 23 - Worksheet P20 Dart Collections

## Introduction

A collection is an object made by grouping together multiple objects (which can be of similar or different data types). This worksheet is designed to familiarize you with two of Dart’s most important collections:

* **Lists**: A collection where the elements are ordered but not necessarily unique. Lists in Dart are very similar to lists in Python.
* **Maps:** The elements are unordered pairs of keys and values. The keys must be unique, but the values can be repeated. Maps in Dart are similar to dictionaries in Python.

As before, we are using [DartPad](https://dartpad.dev?) and if no location for a line of code is specified, assume that we are writing inside the main function:

| **void** main() {  *//TYPE CODE HERE* } |
| --- |

## Lists

Here, we will be demonstrating some basic operations of lists. For more information, we encourage you to visit [the documentation page for the List class](https://api.flutter.dev/flutter/dart-core/List-class.html).

### Defining lists

Suppose we run a coffee shop and need to store the name of people who have purchased their coffees and are waiting in a queue to pick up their drinks. Since queues have an order, we can use lists to represent these people. This is because lists in Dart, similar to lists in Python, are an ordered collection of objects.

Because the names we will store are strings, the data type for our list is: List<String>. Note that the type of objects stored in this list is placed between <> (angled brackets):

| List<String> customers = ['Ştefan', 'Amy', 'jamila', 'Xiu', 'Amy']; |
| --- |

Elements of lists are placed between [] (square brackets) and are separated by , (a comma). Lists’ **elements do not need to be unique**, e.g., customers can have two people called Amy.

### Accessing elements of lists

Let’s tell the first person in the queue that their coffee is ready. We can get elements from lists by placing their index between [] (square brackets) after the name of the list. Recall that the **first element is at index 0**:

| String firstCustomer = customers[0];  print('Your coffee is ready $firstCustomer.'); |
| --- |

Who needs to pick up their coffee? Run the code to find out.

Updating an element in a list is done in the same way. For example, the following code corrects the spelling of Jamilla’s name:

| print('Before: ${customers[2]}'); *// Before: jamila*  customers[2] = 'Jamillah';  print('After: ${customers[2]}'); *// After: Jamillah* |
| --- |

What if we want to alert the last customer? Lists have a length property Since the first person was at index 0, the **last element is at index length – 1** :

| int numberOfCustomers = customers.length;  print('We have $numberOfCustomers customers.'); *// We have 5 customers.*    String lastCustomer = customers[numberOfCustomers - 1];  *// Equally: customers[customers.length - 1] or customers[4]*  print('Sorry $lastCustomer, your coffee is going to be late.'); |
| --- |

### Adding to lists

Elements can be added to lists with [the add method](https://api.dart.dev/stable/2.17.6/dart-core/List/add.html) which works similarly to [the append method in Python](https://docs.python.org/3/tutorial/datastructures.html#more-on-lists). You need to give this function an element to be added to the list and the function appends the element **at the end of the list**.

Here, we have Nadim joining the coffee shop queue:

| customers.add('Nadim');    *// After Nadim has joined, we should update first and last customer*  lastCustomer = customers.last; *// Or: customers[0];*  firstCustomer = customers.first; *// Or: customers[customers.length - 1]*  print('The first customer is $firstCustomer and last one is $lastCustomer');   print(customers); *// [Ştefan, Amy, Jamillah, Xiu, Amy, Nadim]* |
| --- |

We have updated firstCustomer and lastCustomer using the properties [first](https://api.dart.dev/stable/2.17.6/dart-core/List/first.html) and [last](https://api.dart.dev/stable/2.17.6/dart-core/List/last.html) of lists. Which one of these variables has now changed after Nadim joined? Run the code to verify your guess.

You could also use [the insert method](https://api.dart.dev/stable/2.17.6/dart-core/List/insert.html) to add elements but you need to specify where you want the element to be inserted. In the example below Matthew skips the queue and jumps to the second index in the queue:

| customers.insert(1, 'Matthew');  print(customers); *// [Ştefan, Matthew, Amy, Jamillah, Xiu, Amy, Nadim]* |
| --- |

Here is a figure to help illustrate what we have done so far:



Figure 1: Adding to a list

### Removing from lists

Once we have served Ştefan, we can delete him. This can be done using [the remove method](https://api.dart.dev/stable/2.17.6/dart-core/List/remove.html) which takes **an existing element** of the list as its parameter.

| customers.remove('Ştefan');  print(customers); *// [Matthew, Amy, Jamillah, Xiu, Amy, Nadim]* |
| --- |

Notice how the remove method in Dart works similarly to [the remove method in Python](https://docs.python.org/3/tutorial/datastructures.html#more-on-lists).



Figure 2: Removing from a list

Remove “Amy” from customers to find out which one of the two Amys gets removed. Is this behaviour described in the documentation of [the remove method](https://api.dart.dev/stable/2.17.6/dart-core/List/remove.html)?

Save your coffee shop program and open a new DartPad. We will do another example.

Suppose I want to store all the marks that I have got so far in my first year. Marks are given as percentages, so the list needs to store integers, i.e., List<int>. Because there are five modules in the first year, we can fill the list with five zeros at the start:

| List<int> marks = [0, 0, 0, 0, 0];   *// Alternatively: List<int> marks = List<int>.filled(5, 0);*   print('At the start: $marks'); *// At the start: [0, 0, 0, 0, 0]* |
| --- |

You could also use [the filled constructor](https://api.dart.dev/stable/2.17.6/dart-core/List/List.filled.html) to do the same job (shown in the comment).

As the marks get released, we can update marks by changing its values at specific indices:

| marks[0] = 67;  marks[1] = 92;  marks[2] = 54;  marks[3] = 82;  print('First two marks: ${marks[0]}, ${marks[1]}'); *// First two marks: 67, 54*  print('All of them: $marks'); *// All of them: [67, 92, 54, 82, 0]* |
| --- |

Would the line below update the last element of marks? Try it.

| marks[5] = 100; |
| --- |

You will see an error which reminds you that 4 is the last index in a list with 5 elements:

| RangeError (index): Index out of range: index should be less than 5: 5 |
| --- |

### Iterating lists using for loop

We have already been introduced to for loops. Most commonly you will use them to iterate elements of lists. For example, we can use a loop to print all the values in marks:

| **for** (int i = 0; i <= 4; i++) {  print('index: $i, mark: ${marks[i]}');  } |
| --- |

Loops can likewise be used to update lists. For example, we can increase all the marks by 10:

| **for** (int i = 0; i < marks.length; i++) {  *// We can't have more than 100%*  **if** (marks[i] >= 90) {  marks[i] = 100;  } **else** {  marks[i] += 10;  }  }  print('After +10: $marks'); *// After +10: [77, 100, 64, 92, 10]* |
| --- |

Observe how we have used the length of marks as an end for the loop (as opposed to specifying the final index). This is generally a better practice as it works for any list.

### Iterating lists using for-in loop

Another, simpler, type of for loop which you can use to iterate through a list is a for-in loop. Just be careful that you **cannot** **update lists’ content with for-in loops**. Furthermore, you cannot customize when your loop begins, ends and steps (they iterate over all elements, one by one).

For example, to access all the marks and calculate a sum of them, a for-in loop would be a better way of iterating the marks list:



Figure 3: Iterating a list with for-in loop

We first define a variable that is going to hold each mark (in our case this is an integer called mark). Then the keyword in and finally the name of the list to be iterated (in our case, marks). All of this goes between () (the round brackets) of the loop:

Here is the syntax of a for-in loop:

| List<int> marks = [67, 92, 54, 82, 0];   int sum = 0;  **for** (int mark **in** marks) {  sum += mark;  }   double average = sum / marks.length;  print('The average of my marks is: ${average.toStringAsFixed(2)}'); |
| --- |

### Multidimensional lists [Advanced]

Any object can be an element of a list, even lists. This leads us to the concept of multidimensional lists (lists within lists). To demonstrate, let’s define a food diary list.

Each element of foodDiary is a list of food items (as string) consumed within a day of the week. So foodDiary has the type List<List<String>> .

| List<List<String>> foodDiary = [  ['🧇', '🧆', '🥗'], *// Monday*  ['🍳', '🍛'], *// Tuesday*  ['🥯', '🥘', '🥪'], *// Wednesday*  ['🍩', '🌯', '🍲'], *// Thursday*  ['🧇', '🌮', '🍣'], *// Friday*  ['🥞', '🥧', '🍔', '🍟'], *// Saturday*  ['🍪', '🥪', '🍝'] *// Sunday*  ];  print(foodDiary); |
| --- |

There are three meals for most of the days, but there are days with fewer or more (the length of the lists does not need to be the same).

To change my Saturday’s lunch from pies to pizza, I can update the element at index 5 of foodDiary (the sixth list) at index 1 (the second element of that list):

| print('Before: ${foodDiary[5]}'); *// Before: [🥞, 🥧, 🍔, 🍟]*  foodDiary[5][1]='🍕';  print('After: ${foodDiary[5]}'); *// After: [🥞, 🍕, 🍔, 🍟]* |
| --- |

## Maps

The Map collection in Dart are equivalent to [the dictionary in Python](https://docs.python.org/3/tutorial/datastructures.html#dictionaries). A map is an object that contains a set of keys. For every key, a map stores a corresponding value. The **keys are unique** and you can use them to get to the associated values.

Here we will go over the most common operations you can do with maps. For more details visit [the documentation page of the Map class](https://api.flutter.dev/flutter/dart-core/Map-class.html).

### Defining maps

An issue with our coffee shop list is that we cannot store the drinks that the customers have ordered. This is a typical scenario where it would make sense to use maps.

We could find a customer’s drink using their names. So it makes sense to have the names (string) be the keys in our map. The drinks are also stored as strings, so our map is of type Map<String, String> :

| Map<String, String> orders = {  *// Key (name): Value (drink)*  'Ştefan': 'Espresso Frappuccino',  'Amy T': 'Iced Coffee',  'Jamillah': 'Caramel Frappuccino',  'Xiu': 'Caffè Latte',  'Amy E': 'Caramel Frappuccino'  };   print(orders); |
| --- |

Observe that we place the key first, then a : (colon) and afterwards the value. The key and value pairs are separated by commas and placed between curly brackets. Also notice that because the keys must be unique, we have to distinguish between the two customers with the name “Amy” using their surnames.

### Accessing elements of maps

We access the values in maps using the keys. To access a value (in our case someone’s drink), you should specify its key (the customer’s name). Just like how we access elements in lists, the key is placed between [] (square brackets) after the name of the map.

Note that **the key must exist** in the map for us to be able to access or update it:

| print('Ştefan wants: ${orders['Ştefan']}'); *// Ştefan wants: Espresso Frappuccino*  orders['Ştefan'] = 'Iced Chai Tea Latte'; *// Updates Ştefan's drink*  print('Ştefan wants: ${orders['Ştefan']}'); *// Ştefan wants: Iced Chai Tea Latte* |
| --- |

### Adding to maps

Adding keys and values to maps is very similar to accessing elements in them. **If a key does not exist** in a map, the assignment statement adds a new key and value:

| print(orders['Nadim']); *// null as Nadim's order does not exist*  orders['Nadim'] = 'Nitro Cappuccino'; *// Adds Nadim's drink to orders*  print(orders['Nadim']); *// Nitro Cappuccino* |
| --- |

### Removing from maps

Similar to lists, maps provide a [remove method](https://api.flutter.dev/flutter/dart-core/Map/remove.html). All you need to do is provide this function with **an existing key** and it will remove the key and the associated map:

| orders.remove('Ştefan'); *// Removes Ştefan and their drink from orders*  print(orders['Ştefan']); *// null as it no longer exists* |
| --- |

### Iterating maps using for-in loop

Since there is no order in maps, we cannot easily index into them and use a for loop. Instead, we can use a for-in loop. Observe how we must iterate over the keys of the map (accessed using [the keys property of the Map class](https://api.dart.dev/be/180791/dart-core/Map/keys.html)):

| **for** (String k **in** orders.keys) {  print('Key (customer) $k, has value (drink): ${orders[k]}');  } |
| --- |

### Lists as the values in maps [Advanced]

You can technically have maps within maps, but here we will show you how to use lists as the values in maps.

Going back to our food diary example, we can store the day of the week as the key and the corresponding meal’s list as their values:

| Map<String, List<String>> foodDiary = {  'Monday': ['🧇', '🧆', '🥗'],  'Tuesday': ['🍳', '🍛'],  'Wednesday': ['🥯', '🥘', '🥪'],  'Thursday': ['🍩', '🌯', '🍲'],  'Friday': ['🧇', '🌮', '🍣'],  'Saturday': ['🥞', '🥧', '🍔', '🍟'],  'Sunday': ['🍪', '🥪', '🍝']  };  print('Saturday: ${foodDiary['Saturday']}'); *// Saturday: [🥞, 🥧, 🍔, 🍟]*  *// foodDiary has a value for 'Saturday', update the 2nd element in it*  foodDiary['Saturday']![1] = '🍕';   print('Saturday: ${foodDiary['Saturday']}'); *// Saturday: [🥞, 🍕, 🍔, 🍟]* |
| --- |

The ! (exclamation mark) in “foodDiary['Saturday']![1] = '🍕';” is the **null assertion operator**. Without it, the compiler throws an error as it has no guarantee that foodDiary has a value for 'Saturday'. The null assertion operator assures the compiler that there is a value. If we were wrong, there was no value (null), and we get an error when we run our code.

## Worked example

There are still websites that store passwords as [ASCII characters](https://www.asciitable.com) as opposed to [Unicode](https://unicode-table.com/en/). The ASCII table has 27 – 1 = 127 characters. This is because each character is stored using 8 bits and the most significant is not used. Even with the 8th bit, we get 27 – 1 = 255 characters in ASCII. In comparison, Unicode (at the moment) has 144,697 characters. Clearly, the more characters you have, the harder it is to guess passwords.

One way to make it more difficult to guess ASCII passwords is to generate random ones. This way the hackers cannot guess a password by trying commonly used phrases and instead need to try every combination of characters (a Brute-force attack). Our challenge here is to write a random password generator in Dart.

We have already seen how to generate random numbers in Dart:

| **import** 'dart:math';  **void** main() {  int randomNumber = random1to100();  print('Random number is: $randomNumber'); }  */// Returns a random number from 0 to 100* int random1to100() {  Random randomGenerator = Random();  **return** randomGenerator.nextInt(101); } |
| --- |

The string class in Dart provides [the constructor String.fromCharCode](https://api.dart.dev/stable/2.17.3/dart-core/String/String.fromCharCode.html) allowing you to create a character given a number. This number can be an index from the [ASCII table](https://www.asciitable.com):

| String character = String.fromCharCode(122);  print(character); *// z* |
| --- |

Start by writing a new function called randomIndex that returns a random index from the [ASCII table](https://www.asciitable.com) (not the extended one). Make sure that the index corresponds to a printable character, e.g., avoid index 8, which corresponds to a backspace. If you are stuck, use the keyboard shortcut F1 on DartPad and click on nextInt to read [the documentation of nextInt method](https://api.dart.dev/stable/2.17.3/dart-math/Random/nextInt.html).

Next, write a function randomIndices that accepts an integer as its input and returns a list of that size filled with random indices. Remember that randomIndices should use randomIndex. The function’s signature is therefore “List<int> randomIndices(int)” and as an example, randomIndices(5) could return “[51, 32, 123, 54, 110]”.

Finally, write a function randomString that accepts an integer as its input and returns a randomly generated string of the specified size. randomString should call randomIndices to generate a list of indices first and then loop over them (ideally with a for-in loop). For example, randomString(16) could return “OqoR/! Yg3(R!)}'”.

You do not need to place the functions in any specific order, it will work nonetheless. Once you are done, compare your work with other students before looking at our model answer at [the end of this document](#_lgokib4neefj).

## Programming exercises

Test your work for each question by calling it in main, and make sure to format and save it.

1. Write a listInfo function that takes a list of strings ([similar to customers](#_eh9c7wk8dh97)) and prints its length, first and last element.

Start with the following snippet and write your code where specified by the comments:

| **void** main() {  List<String> customers = ['Ştefan', 'Amy', 'Jamilla', 'Xiu', 'Amy'];  listInfo(customers); *// Size=5, first=Ştefan, last=Amy*    customers.add('Nadim');  listInfo(customers); *// Size=6, first=Ştefan, last=Nadim* }  **void** listInfo(List<String> list){  *// WRITE YOUR CODE HERE* } |
| --- |

1. Write a function capMarks that accepts a list of integers representing assessment marks, and caps them to 40. Copy and complete the following snippet as your starting point:

| **void** main() {  List<int> myMarks = [47, 65, 34, 73, 0];  capMarks(myMarks);  print(myMarks); *// Should print: [40, 40, 34, 40, 0]* }  void capMarks(List<int> marks) {  *// WRITE YOUR CODE HERE* } |
| --- |

1. Copy the code that constructs the orders map from [earlier in the worksheet](#_e1sd4st3wn3u) to your main function. Then, write a for-in loop that iterates through the keys of orders and prints the name of every customer with their drink. Your output should resemble the following:

| Ştefan has ordered: Espresso Frappuccino  Amy T has ordered: Iced Coffee  Jamillah has ordered: Caramel Frappuccino  Xiu has ordered: Caffè Latte  Amy E has ordered: Caramel Frappuccino |
| --- |

1. Write a function average that takes a list of integers as its parameter and returns their average. Since the average of integers may not necessarily be a whole number, the signature of this function is “double average(List<int>)”. For example average([47, 65, 34, 73, 0]) should return 43.8.
2. Inside main, define a map called credits. The keys should be your module names and the values should be the credits for each module (as an integer). Therefore, the type of credit is “Map<String, int>”. Test your map and print it similar to the output shown below:

| Architecture and Operating Systems has 20 credits  Programming has 40 credits  Core Computing Concepts has 20 credits |
| --- |

If you are unsure about your module credits, use [this page](https://course-module-catalog.port.ac.uk/#/) to search for your course. Once on the course page, change “Module” to “course” and search for your course name.

1. [Hard]: Similar to the previous question, write a map called marksMap in main. The values corresponding to each key should be a list of all the marks that you have received for that given module. Make sure that the keys are identical to the credits map you defined in the previous question. Your program should output something like this:

| In Architecture and Operating Systems I got: 67, 61, 54, 82  In Programming I got: 67, 92, 54, 82, 34, 56, 38, 40  In Core Computing Concepts I got: 67, 32, 54, 82 |
| --- |

Note: You may get this error message when you iterate a map where the values are a list: “A nullable expression can't be used as an iterator in a for-in loop”. This is because the compiler cannot guarantee that the value corresponding to the key is not null. You need to use [the null assertion operator](https://dart.dev/codelabs/null-safety#:~:text=The%20null%20assertion%20operator%20(!),-If%20you're&text=just%20after%20the%20expression%2C%20you,an%20exception%20at%20run-time.) (!) to tell the compiler that you believe the value will never be null. [The documentation page for this error](https://dart.dev/tools/diagnostic-messages?utm_source=dartdev&utm_medium=redir&utm_id=diagcode&utm_content=unchecked_use_of_nullable_value#unchecked_use_of_nullable_value) provides an example.

1. [Hard]: Write a function gradeCalculator which takes two maps with the same signature as credits and marksMap. gradeCalculator should first calculate the average of all marks for every module (we assume that all assessments in a module have the same weight). Then gradeCalculator should take the average of the produced values to calculate the overall mark for a year of studies. Finally, gradeCalculator should return a grade from the weighted average of the module marks as a String: “1st” for +70%, “2:1” for +60%, “2:2” for 50%, “3rd” for +40% and “fail” for any lower value.

For example, given the credits and module marks from the previous two questions, gradeCalculator would return “2:1”. This is because the averages of the modules are 66.00, 57.88 and 58.75 respectively. Therefore the overall mark is 60.13 (a 2.1).

## Answer to the worked example

Check out our solution to the question opened in [the worked example section](#_7ykuk273e4nj) by visiting [this DartPad](https://dartpad.dev/97b2916b886636575d06a958b90952bd).

Observe that for every function, we have an example in the block of comments. This is a great way to remind yourself or tell others how your functions work. See for example:

| */// Returns a random ASCII index corresponding to a printable character* *///* */// ```dart* */// int index = randomIndex();* */// print(index); // 50* */// ```* int randomIndex() {  int min = 32;  int max = 126;   Random randomGenerator = Random();  **return** randomGenerator.nextInt(max - min) + min; } |
| --- |

You can see the documentation of randomIndex, defined above, by first using the documentation shortcut in DartPad (F1) and then clicking on the name of the function. This should display the following in the documentation panel (bottom right panel of DartPad):

