# Lab 05

Instructor: Sidra Khatoon E-mail: skhatoon[@uit.edu](mailto:ad@uit.edu)

# Objective

The purpose of this lab session is to practice Flutter tool. Create a simple application on flutter.

**Student Information**

|  |  |
| --- | --- |
| **Student Name** |  |
| **Student ID** |  |
| **Date** |  |

**Assessment**

|  |  |
| --- | --- |
| **Marks Obtained** |  |
| **Remarks** |  |
| **Signature** |  |

# Objective

The purpose of this lab session is to practice Dart language.

# Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit?

Submit lab work using Teams.

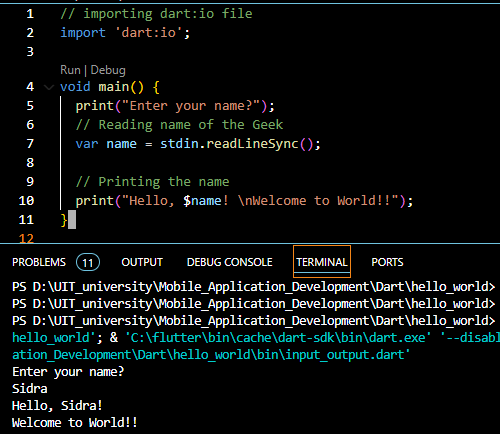
**Dart – Standard Input and Output:**

**Standard Input in Dart:**

In Dart programming language, you can take standard input from the user through the console via the use of readLineSync() function. To take input from the console you need to import a library, named dart:io from libraries of Dart.

**About Stdin Class:**

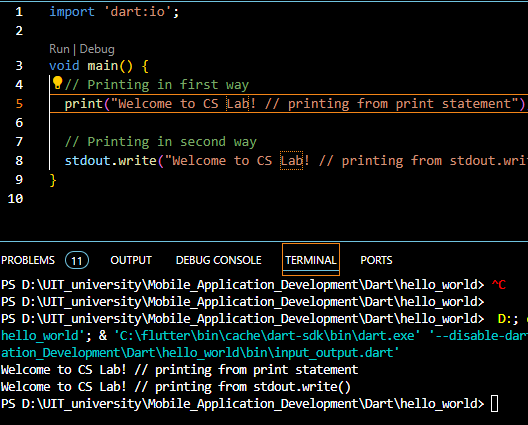
This class allows the user to read data from standard input in both synchronous and asynchronous ways. The method readLineSync() is one of the methods used to take input from the user. Refer to the official doc for other methods.



**Standard Output in Dart:**

In dart, there are two ways to display output in the console:

1. Using print statement.
2. Using stdout.write() statement.

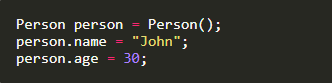
****

**Constructor in Dart**

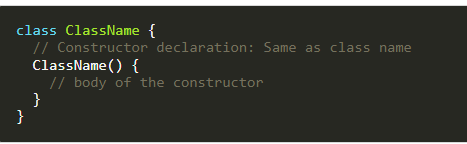
A constructor is a special method used to initialize an object. It is called automatically when an object is created, and it can be used to set the initial values for the object’s properties. For example, the following code creates a Person class object and sets the initial values for the name and age properties.



If you don’t define a constructor for class, then you need to set the values of the properties manually. For example, the following code creates a Person class object and sets the values for the name and age properties.

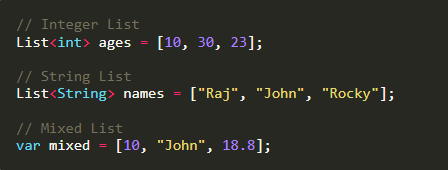


* The constructor’s name should be the same as the class name.
* Constructor doesn’t have any return type.

****

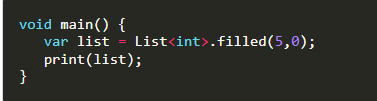
**List in Dart**

If you want to store multiple values in the same variable, you can use List. List in dart is similar to Arrays in other programming languages. E.g. to store the names of multiple students, you can use a List. The List is represented by Square Braces []. You can create a List by specifying the initial elements in a square bracket. Square bracket [] is used to represent a List.

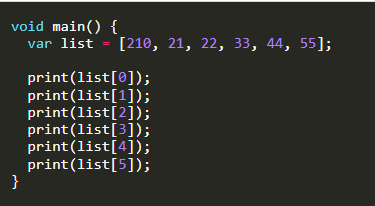


**Types of Lists:**

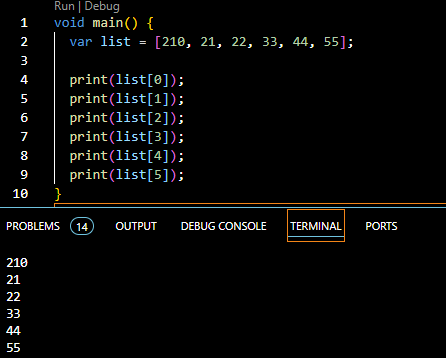
* Fixed Length List: The fixed-length Lists are defined with the specified length. You cannot change the size at runtime. This will create List of 5 integers with the value 0.



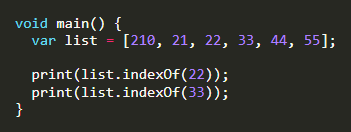
* Growable List: A List defined without a specified length is called Growable List. The length of the growable List can be changed in runtime.



You can access the List item by index. Remember that the List index always starts with 0.



You can also get the index by value.

****

**List Properties in Dart:**

* **first**: It returns the first element in the List.
* **last**: It returns the last element in the List.
* **isEmpty**: It returns **true** if the List is empty and **false** if the List is not empty.
* **isNotEmpty**: It returns **true** if the List is not empty and **false** if the List is empty.
* **length**: It returns the length of the List.
* **reversed**: It returns a List in reverse order.
* **single**: It is used to check if the List has only one element and returns it.

**Adding Item to List:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | | **Method** | **Description** | | --- | --- | | **add()** | Add one element at a time and returns the modified List object. | | **addAll()** | Insert the multiple values to the given List, and each value is separated by the commas and enclosed with  a square bracket ([]). | | **insert()** | Provides the facility to insert an element at a specified index position. | | **insertAll()** | Insert the multiple value at the specified index position. | | | |

**Removing List Elements:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Method** | **Description** | | --- | --- | | **remove()** | Removes one element at a time from the given List. | | **removeAt()** | Removes an element from the specified index position and returns it. | | **removeLast()** | Remove the last element from the given List. | | **removeRange()** | Removes the item within the specified range. | |

**Dart Sets**

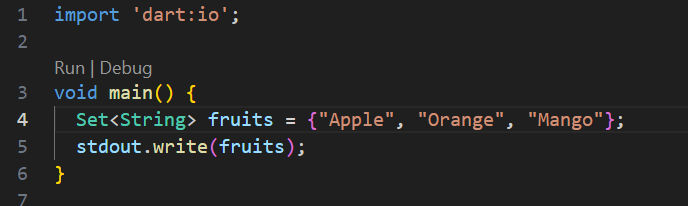
Set is a unique collection of items. You cannot store duplicate values in the Set. It is unordered, so it can be faster than lists while working with a large amount of data. Set is useful when you need to store unique values without considering the order of the input. E.g., fruits name, months name, days name, etc. It is represented by **Curley Braces {}.**

**Syntax:**

****

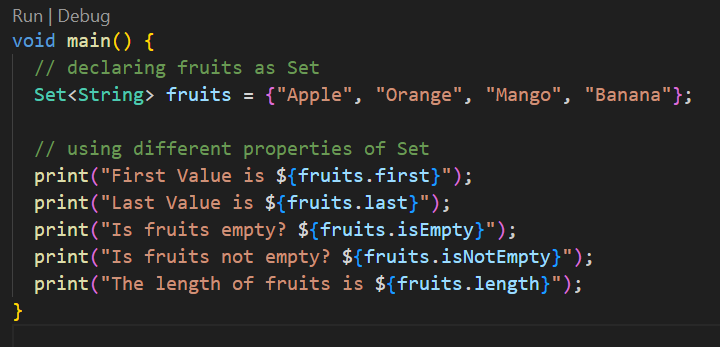
**How To Create A Set In Dart:**

You can create a Set in Dart using the Set type annotation. Here Set<String> means only text is allowed in the Set.



**Set Properties In Dart:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Properties** | **Work** | | --- | --- | | **first** | To get first value of Set. | | **last** | To get last value of Set. | | **isEmpty** | Return true or false. | | **isNotEmpty** | Return true or false. | | **length** | It returns the length of the Set. | |

****

**Check The Available Value:**

If you want to see whether the Set contains specific items or not, you can use the **contains** method, which returns true or false.

****

**Add & Remove Items In Set:**

you can add or remove items in a Set. To add items use **add()** method and to remove use **remove()** method.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Method** | **Description** | | --- | --- | | **add()** | Add one element to Set. | | **remove()** | Removes one element from Set. | | **addAll()** | Insert the multiple values to the given Set. | |



**Set Methods in Dart:**

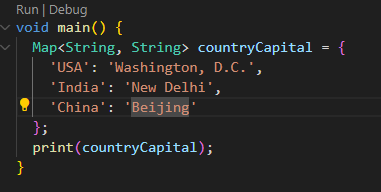
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Method** | **Description** | | --- | --- | | **clear()** | Removes all elements from the Set. | | **difference()** | Creates a new Set with the elements of this that are not in other. | | **elementAt()** | Returns the index value of element. | | **intersection()** | Find common elements in two sets. | |

**Map in Dart**

In a Map, data is stored as keys and values. In Map, each key must be unique. They are similar to HashMaps and Dictionaries in other languages.

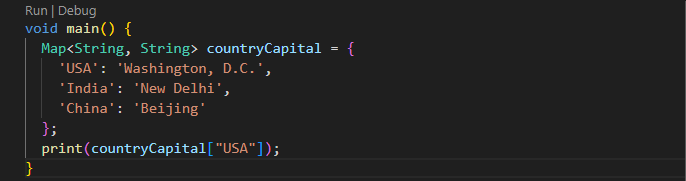
**How To Create Map in Dart:**

we are creating a Map for String and String. It means keys and values must be the type of String. You can create a Map of any kind as you like.



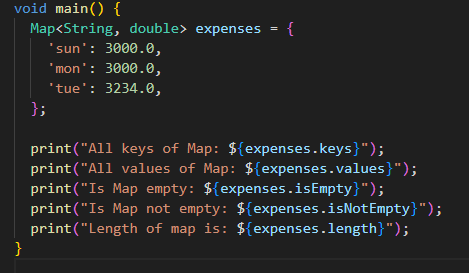
**Access Value from Key:**

You can find the value of Map from its key. Here we are printing Washington, D.C. by its key, i.e., USA.

****

**Map Properties in Dart:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Properties** | **Work** | | --- | --- | | **keys** | To get all keys. | | **values** | To get all values. | | **isEmpty** | Return true or false. | | **isNotEmpty** | Return true or false. | | **length** | It returns the length of the Map. | |

****

**Map Methods In Dart:**

Some useful Map methods in dart

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Properties** | **Work** | | --- | --- | | **keys.toList()** | Convert all Maps keys to List. | | **values.toList()** | Convert all Maps values to List. | | **containsKey(‘key’)** | Return true or false. | | **containsValue(‘value’)** | Return true or false. | | **clear()** | Removes all elements from the Map. | | **removeWhere()** | Removes all elements from the Map if condition is valid. | |

**Asynchronous Programming in Dart**

Asynchronous Programming is a way of writing code that allows a program to do multiple tasks at the same time. Time consuming operations like fetching data from the internet, writing to a database, reading from a file, and downloading a file can be performed without blocking the main thread of execution.

**Synchronous Programming:**

In Synchronous programming, the program is executed line by line, one at a time. Synchronous operation means a task that needs to be solved before proceeding to the next one.

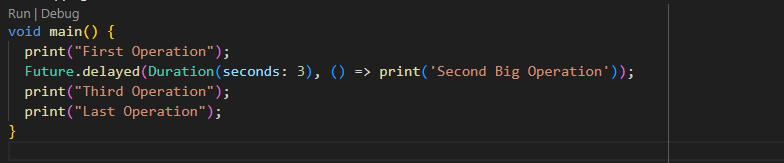
A screen shot of a computer program

Description automatically generated

Here in this example, you can see that it will print line by line. Let’s suppose Second Big Operation takes 3 seconds to load then Third Operation and Last Operation need to wait for 3 seconds. To solve this issue asynchronous programming is introduced.

**Synchronous Programming:**

In Asynchronous programming, program execution continues to the next line without waiting to complete other work. It simply means, **Don’t wait**. It represents the task that doesn’t need to solve before proceeding to the next one.



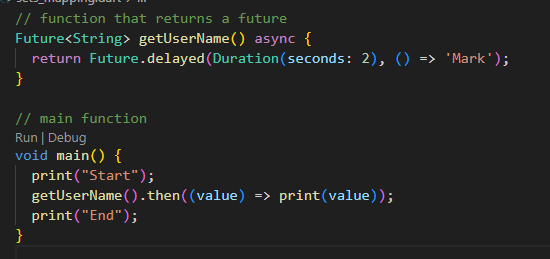
Here in this example, you can see that it will print **Second Big Operation** at last. It is taking 3 seconds to load and **Third Operation** and **Last Operation** don’t need to wait for 3 seconds. This is the problem solved by Asynchronous Programming.

**Future In Dart:**

In dart, the Future represents a value or error that is not yet available. It is used to represent a potential value, or error, that will be available at some time in the future. You can create a future in dart by using **Future** class. Here the function will return **Future<String>** after 5 seconds.

**How To Use Future In Dart:**

We can use future in dart by using **then()** method. Here the function will return **Future<String>** after 5 seconds.



Future represents the result of an asynchronous operation and can have 2 states.

State Of Future

* Uncompleted
* Completed

**Uncompleted:**

When you call an asynchronous function, it returns to an uncompleted future. It means the future is waiting for the function asynchronous operation to finish or to throw an error.

**Completed:**

It can be completed with value or completed with error. Future<int> produces an int value, and Future<String> produces a String value. If the future doesn’t produce any value, then the type of future is Future<void>.

**Async And Await In Dart:**

Async/await is a feature in Dart that allows us to write asynchronous code that looks and behaves like synchronous code, making it easier to read.

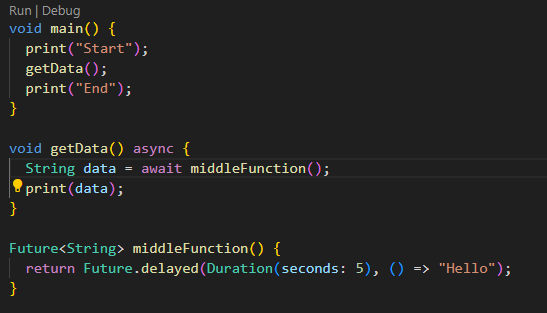
When a function is marked async, it signifies that it will carry out some work that could take some time and will return a Future object that wraps the result of that work.

The await keyword, on the other hand, allows you to delay the execution of an async function until the awaited Future has finished. This enables us to create code that appears to be synchronous but is actually asynchronous.

The async and await keywords both provide a declarative way to define an asynchronous function and use their results. You can use the async keyword before a function body to make it asynchronous. You can use the await keyword to get the completed result of an asynchronous expression.

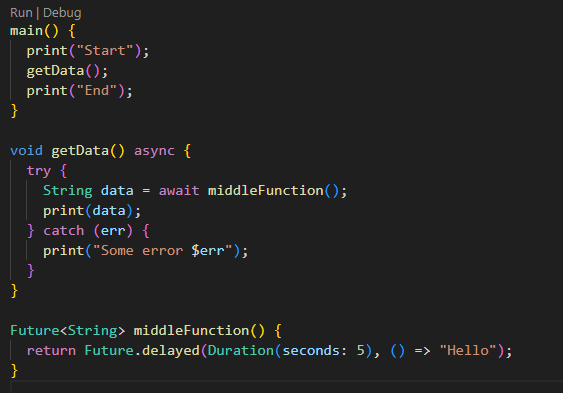
**Important Concept:**

* To define an Asynchronous function, add async before the function body.
* The await keyword work only in the async function.

****

**Handling Errors**

You can handle errors in the dart async function by using try-catch. You can write try-catch code the same way you write synchronous code.



In the above example, try-catch handles the exception that could come after the program is executed.

**Assessments**

**Submit any 6 program in lab assessment:**

1. Create a list of names and print all names using list.
2. Create a program that’s reads list of expenses amount using user input and print total.
3. Create an empty list of type string called days. Use the add method to add names of 7 days and print all days.
4. Add your 7 friend names to the list. Use where to find a name that starts with alphabet a.
5. Create a map with name, address, age, country keys and store values to it. Update country name to other country and print all keys and values.
6. Create a map with name, phone keys and store some values to it. Use where to find all keys that have length 4.
7. Create a simple to-do application that allows user to add, remove, and view their task.
8. Write a program in dart that uses Future class to perform multiple asynchronous operations, wait for all of them to complete, and then print the results.
9. Write a Dart program to calculate the sum of two numbers using async/await.
10. Write a Dart program that takes in two integers as input, waits for 3 seconds, and then prints the sum of the two numbers.
11. Write a Dart program that takes a list of strings as input, sorts the list asynchronously, and then prints the sorted list.
12. Write a Dart program that takes a list of integers as input, multiplies each integer by 2 asynchronously, and then prints the modified list.
13. Write a Dart program that takes a string as input, reverses the string asynchronously, and then prints the reversed string.