

**SEB-312 Mobile Application Development**

**LAB # 02**

**LAB Title**

|  |
| --- |
| Writing simple Dart programs: Variables, loops, and functions, implementing basic OOP concepts: Creating classes, objects, and methods., Manipulating Lists, Sets, and Maps, implementing asynchronous functions with Future and async/await, |

**Assessment of CLO: 03, PLO: 05**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** |  | | |
| **Roll No.** |  | | |
| **Semester** |  | **Session** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Perf. Level**  **Criteria** | **Excellent**  **(2.5)** | **Good**  **(2)** | **Satisfactory**  **(1.5)** | **Needs Improvement**  **(0 ~ 1)** | **Marks Obtained** |
| **1** | Project Execution & Implementation | Fully functional, optimized, and well-structured. | Minor errors, mostly functional. | Some errors, requires guidance. | Major errors, non-functional, or not Performed. |  |
| **2** | Results & Debugging  Or Troubleshooting | Accurate results with effective debugging  Or Troubleshooting. | Mostly correct, some debugging Or Troubleshooting needed. | Partial results, minimal debugging  Or Troubleshooting. | Incorrect results, no debugging Or Troubleshooting, or not attempted. |  |
| **3** | Problem-Solving & Adaptability  (VIVA) | Creative approach, efficiently solves challenges. | Adapts well, minor struggles. | Some adaptability, needs guidance. | Lacks innovation or no innovation, unable to solve problems. |  |
| **4** | Report Quality & Documentation | Clear, structured, with detailed visuals. | Mostly clear, minor gaps. | Some clarity issues, missing details. | Poorly structured, lacks clarity, or not submitted. |  |
| **Total Marks Obtained Out of 10** | | | | | |  |

**Experiment evaluated by**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor’s Name** | **Sidra Khatoon** | | |
| **Date** |  | **Signature** |  |

**Objective**

The objective of this lab session is to practice Dart language

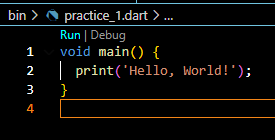
**Instructions**

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.

**DART Language:**

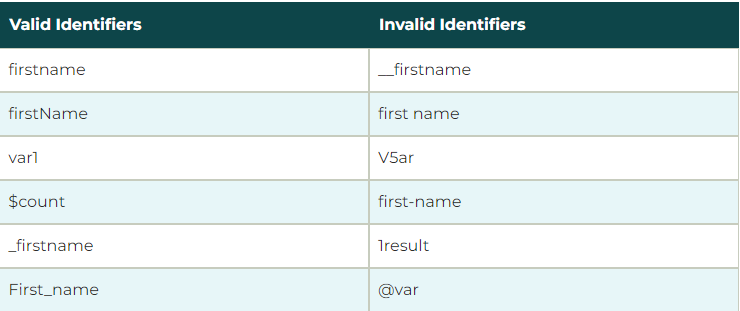
**Hello World:**

Every app requires the top-level main() function, where execution starts. Functions that don't explicitly return a value have the void return type. To display text on the console, you can use the top-level print() function:



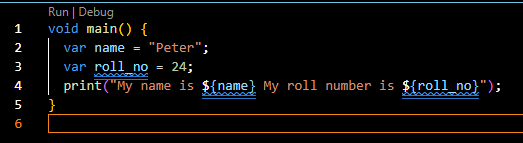
**Dart Identifiers:**

1. Identifiers are the name which is used to define variables, methods, class, and function, etc. An Identifier is a sequence of the letters ([A to Z], [a to z]), digits ([0-9]) and underscore(\_), but remember that the first character should not be a numeric.
2. The first character should not be a digit.
3. Special characters are not allowed except underscore (\_) or a dollar sign ($).
4. Two successive underscores (\_\_) are not allowed.
5. The first character must be alphabet (uppercase or lowercase) or underscore.
6. Identifiers must be unique and cannot contain whitespace.
7. They are case sensitive. The variable name Joseph and joseph will be treated differently.



**Dart Printing and String Interpolation**

The print() function is used to print output on the console, and $expression is used for the string interpolation. Below is an example.



**Semicolon in Dart**

The semicolon is used to terminate the statement that means, it indicates the statement is ended here. It is mandatory that each statement should be terminated with a semicolon (;). We can write multiple statements in a single line by using a semicolon as a delimiter. The compiler will generate an error if it is not use properly.

**Dart Whitespace and Line Breaks**

The Dart compiler ignores whitespaces. It is used to specify space, tabs, and newline characters in our program. It separates one part of any statement from another part of the statement. We can also use space and tabs in our program to define indentation

**Block in Dart**

The block is the collection of the statement enclosed in the curly braces. In Dart, we use curly braces to group all of the statements in the block. Consider the following syntax. And provide the proper format for the program. It makes code easy to understand and readable.

**Dart Datatypes**

The data types are the most important fundamental features of programing language. In Dart, the data type of the variable is defined by its value. The variables are used to store values and reserve the memory location. The data-type specifies what type of value will be stored by the variable. Each variable has its data-type. The Dart is a static type of language, which means that the variables cannot modify

**Dart Number**

The Darts Number is used to store the numeric values. The number can be two types - integer and double.

**Integer -** Integer values represent the whole number or non-fractional values. An integer data type represents the 64-bit non-decimal numbers between -263 to 263. A variable can store an unsigned or signed integer value. The example is given below –

int marks = 80;

**Double -** Double value represents the 64-bit of information (double-precision) for floating number or number with the large decimal points. The double keyword is used to declare the double type variable.

double pi = 3.14;

**Dart Strings**

A string is the sequence of the character. If we store the data like - name, address, special character, etc. It is signified by using either single quotes or double quotes. A Dart string is a sequence of UTF-16 code units.

var msg = "Welcome to JavaTpoint";

**Dart Boolean**

The Boolean type represents the two values - true and false. The bool keyword uses to denote Boolean Type. The numeric values 1 and 0 cannot be used to represent the true or false value.

bool isValid = true;

**Dart Lists**

In Dart, The list is a collection of the ordered objects (value). The concept of list is similar to an array. An array is defined as a collection of the multiple elements in a single variable. The elements in the list are separated by the comma enclosed in the square bracket[]. The sample list is given below.

var list = [1,2,3]

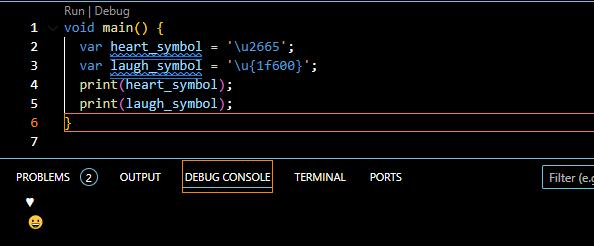
**Dart Maps**

The maps type is used to store values in key-value pairs. Each key is associated with its value. The key and value can be any type. In Map, the key must be unique, but a value can occur multiple times. The Map is defined by using curly braces ({}), and comma separates each pair.

var student = {'name': 'Joseph', 'age':25, 'Branch': 'Computer Science'}

**Dart Runes**

As we know that, the strings are the sequence of Unicode UTF-16 code units. Unicode is a technique which is used to describe a unique numeric value for each digit, letter, and symbol. Since Dart Runes are the special string of Unicode UTF-32 units. It is used to represent the special syntax.



**Dart Dynamic Type**

Dart is an optionally typed language. If the variable type is not specified explicitly, then the variable type is dynamic. The dynamic keyword is used for type annotation explicitly.

**Dart Variable**

Variable is used to store the value and refer the memory location in computer memory. When we create a variable, the Dart compiler allocates some space in memory. The size of the memory block of memory is depended upon the type of variable. To create a variable, we should follow certain rules. Here is an example of a creating variable and assigning value to it.

var name = 'Devansh';

Here the variable called **name** that holds 'Devansh' string value. In [Dart](https://www.javatpoint.com/dart-programming), the variables store references. The above variable stores reference to a String with a value of Devansh.

**Rule to Create Variable**

Creating a variable with a proper name is an essential task in any programming language. The Dart has some rules to define a variable. These rules are given below.

1. The variable cannot contain special characters such as whitespace, mathematical symbol, runes, Unicode character, and keywords.
2. The first character of the variable should be an alphabet([A to Z],[a to z]). Digits are not allowed as the first character.
3. Variables are case sensitive. For example, - variable age and AGE are treated differently.
4. The special character such as #, @, ^, &, \* are not allowed expect the underscore(\_) and the dollar sign($).
5. The variable name should be retable to the program and readable.

**How to Declare Variable in Dart?**

We need to declare a variable before using it in a program. In Dart, The **var** keyword is used to declare a variable. The Dart compiler automatically knows the type of data based on the assigned to the variable because Dart is an infer type language. The syntax is given below.

var <variable\_name>  = <value>;

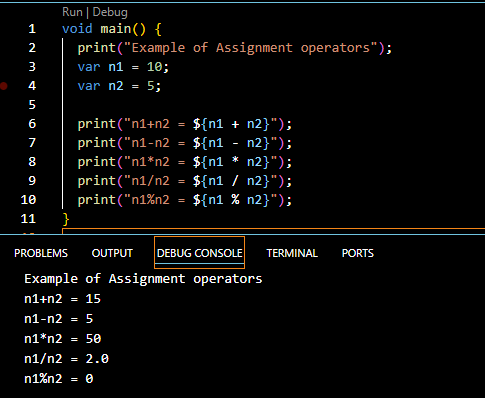
or

var <variable\_name>;

**Operators**

Dart supports the all operators that are present any programming language. You can implement any of  [operator as class members](https://dart.dev/language/methods#operators).

**Arithmetic Operators**



**Increment and Decrement**

++ and -- operators are known as increment and decrement operators and also known as unary operators, respectively. Unary operators, operate on single operand where ++ adds 1 to operands and -- subtract 1 to operand respectively. The unary operators can be used in two ways – postfix and prefix. If ++ is used as a postfix(like x++), it returns the value

of operand first then increments the value of x. If – is used as a prefix(like ++x), it increases the value of x.

**Assignment Operator**

=

+=

-= ,

\*=

~/= ,

%=

**Relational Operators**

==

!=

,<

>

<=

>=  
**Bitwise Operators**

AND &

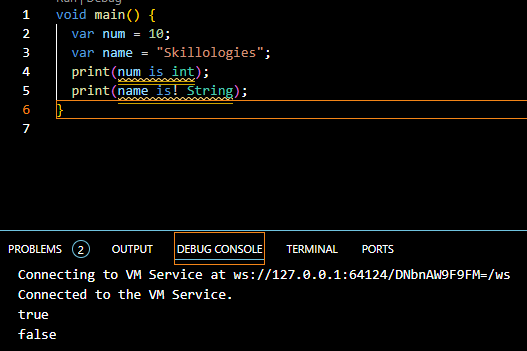
OR | |

**Type Test Operators**

as – It is used for typecast.

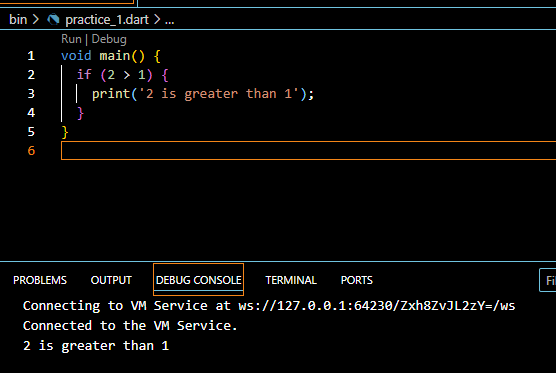
is – It returns TRUE if the object has specified type.

is! – It returns TRUE if the object has not specified type.

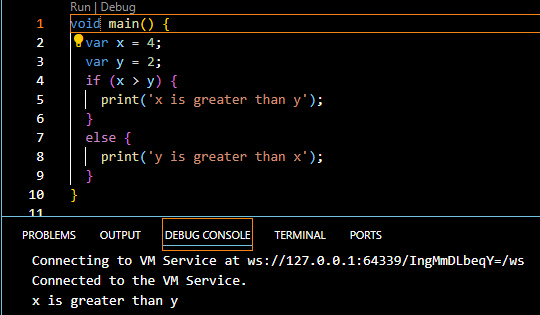


**Control Flow Statements**

**If statement:**



**Else-if statement:**

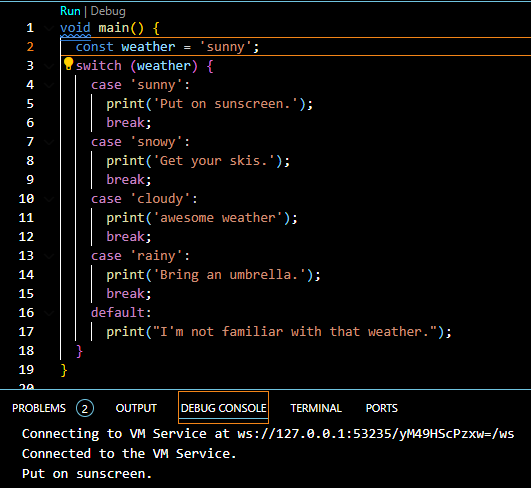


**If-else-if statement:**



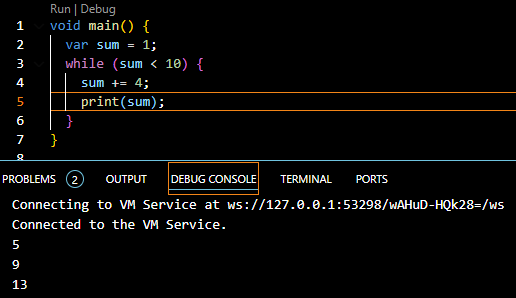
**Switch Statements**

An alternate way to handle control flow, especially for multiple conditions, is with a switch statement.



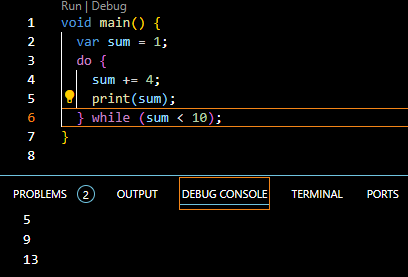
**While loop:**

A while loop repeats a block of code as long as a Boolean condition is true.



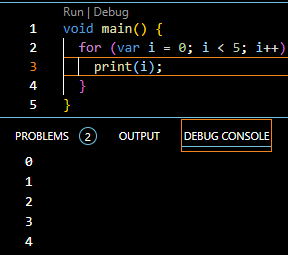
**DO-while loop:**

A variant of the while loop is called the do-while loop. It differs from the while loop in that the condition is evaluated at the *end* of the loop rather than at the beginning. Thus, the body of a do-while loop is always executed at least once.



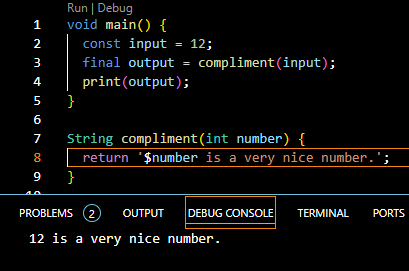
For loop:

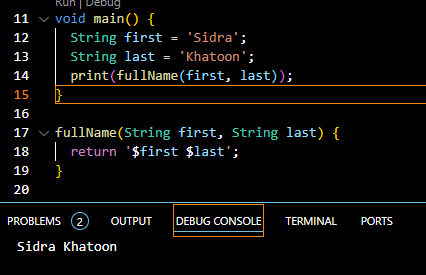
For loop is probably the most common loop you’ll see, and you use it to run a block of code a set number of times.



**Function:**

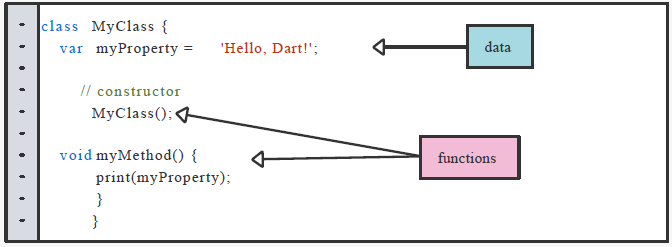
A function is one small task, or sometimes a collection of several related tasks, that you can use in conjunction with other functions to accomplish a larger task. In Dart, a function consists of a return type, a name, a parameter list in parentheses and a body enclosed in braces.



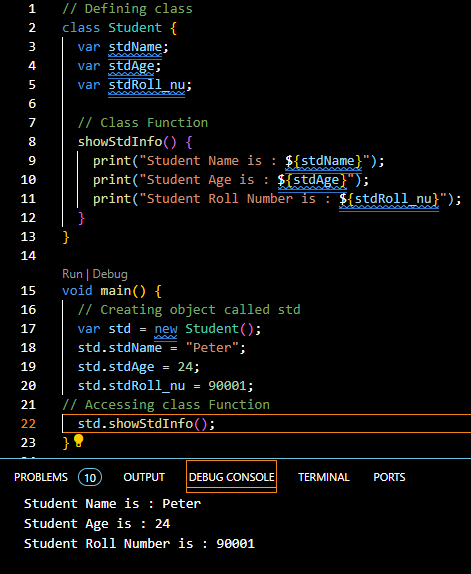
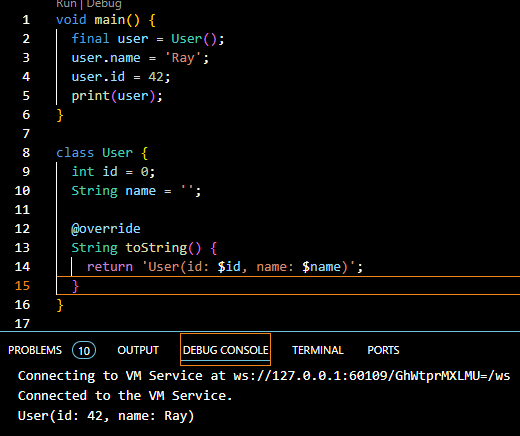


**Classes**

Classes are a core component of object-oriented programming. They’re used to combine data and functions inside a single structure.



The functions exist to transform the data. Functions inside a class are known as methods, whereas constructors are special methods you use to create objects from the class.



**Encapsulation:**

Encapsulation is one of the important concepts of object-oriented programming. In Dart, Encapsulation means hiding data within a library, preventing it from outside factors. It helps you control your program and prevent it from becoming too complicated.

**Encapsulation can be achieved by:**

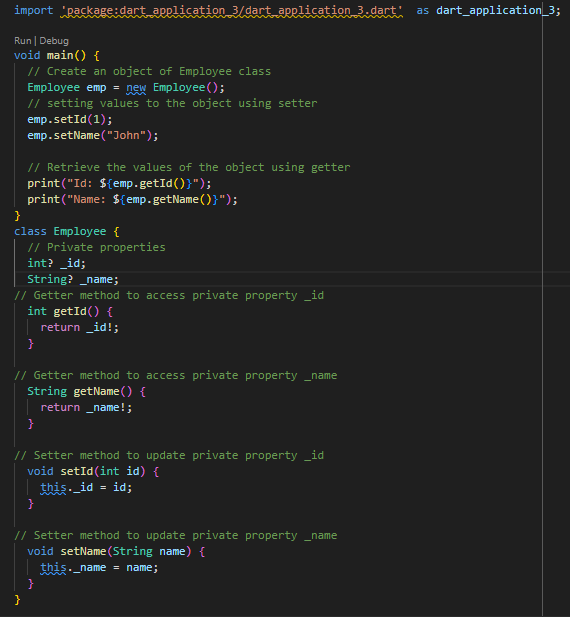
Declaring the class properties as private by using underscore (\_).

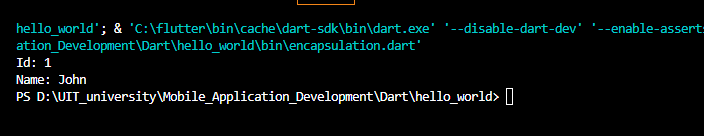
Providing public getter and setter methods to access and update the value of private property.

**Getter and Setter Methods:**

Getter and setter methods are used to access and update the value of private property. Getter methods are used to access the value of private property. Setter methods are used to update the value of private property.

For example, we will create a class named Employee. The class will have two private properties \_id and \_name. We will also create two public methods getId() and getName() to access the private properties. We will also create two public methods setId() and setName() to update the private properties.





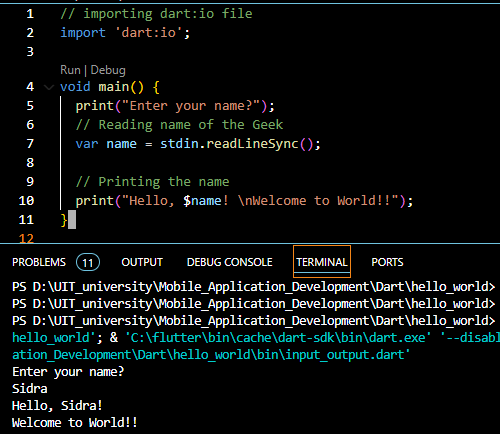
**Dart – Standard Input and Output:**

**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:**

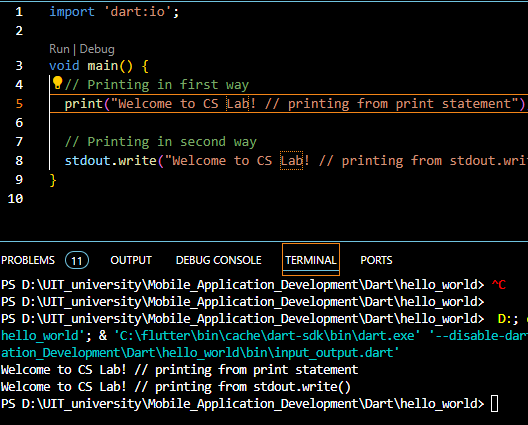
This class allows the user to read data from standard input in both synchronous and asynchronous ways. The method readLineSync() is one of the methods used to take input from the user. Refer to the official doc for other methods.



**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.

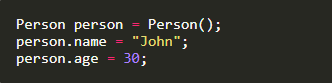
****

**Constructor in Dart**

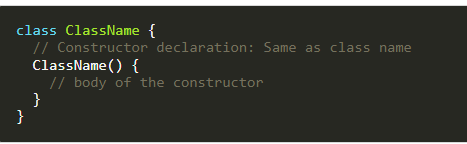
A constructor is a special method used to initialize an object. It is called automatically when an object is created, and it can be used to set the initial values for the object’s properties. For example, the following code creates a Person class object and sets the initial values for the name and age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

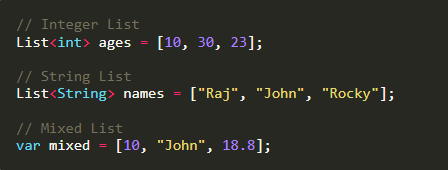


* The constructor’s name should be the same as the class name.
* Constructor doesn’t have any return type.

****

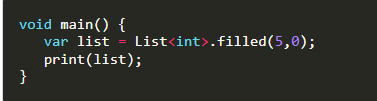
**List in Dart**

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.

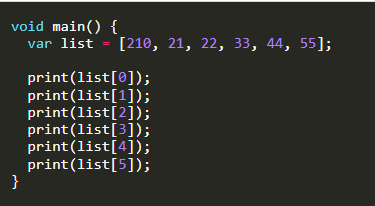


**Types of Lists:**

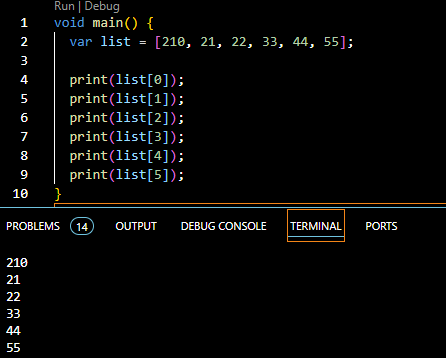
* Fixed Length List: The fixed-length Lists are defined with the specified length. You cannot change the size at runtime. This will create List of 5 integers with the value 0.



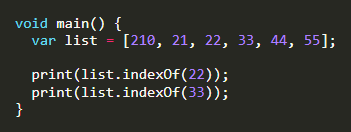
* Growable List: A List defined without a specified length is called Growable List. The length of the growable List can be changed in runtime.



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**

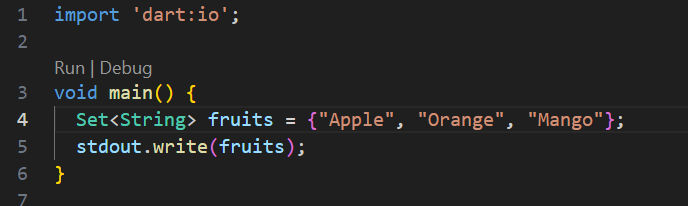
Set is a unique collection of items. You cannot store duplicate values in the Set. It is unordered, so it can be faster than lists while working with a large amount of data. Set is useful when you need to store unique values without considering the order of the input. E.g., fruits name, months name,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

**Syntax:**

****

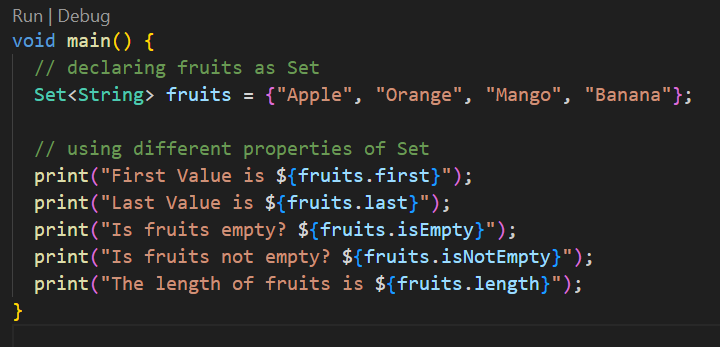
**How to Create A Set In Dart:**

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:**

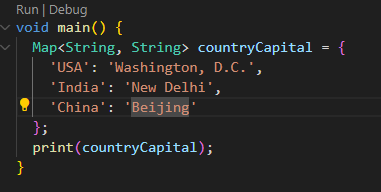
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Method** | **Description** | | --- | --- | | **clear()** | Removes all elements from the Set. | | **difference()** | Creates a new Set with the elements of this that are not in other. | | **elementAt()** | 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.

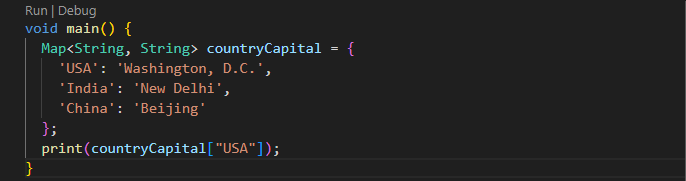
**How To Create Map in Dart:**

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.



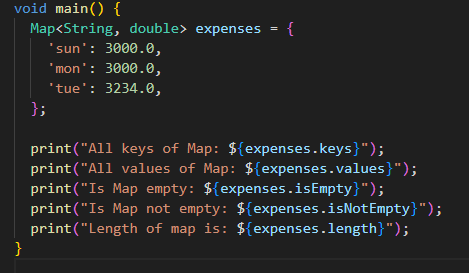
**Access Value from Key:**

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.

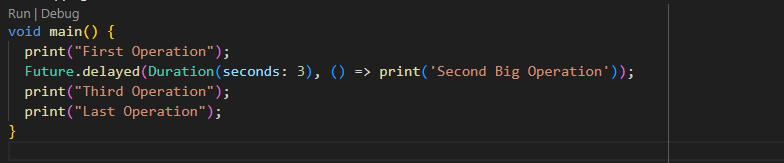
A screen shot of a computer program

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.



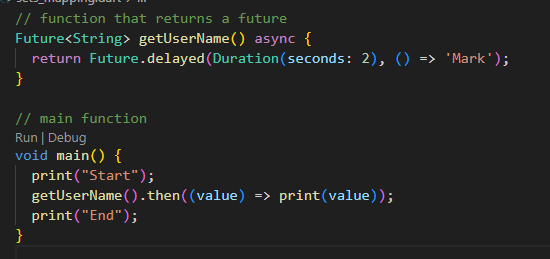
Here in this example, you can see that it will print **Second Big Operation** at last. It is taking 3 seconds to load and **Third Operation** and **Last Operation** don’t need to wait for 3 seconds. This is the problem solved by Asynchronous Programming.

**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.

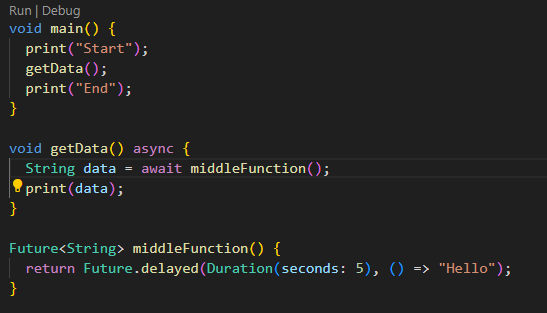
When a function is marked async, it signifies that it will carry out some work that could take some time and will return a Future object that wraps the result of that work.

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.

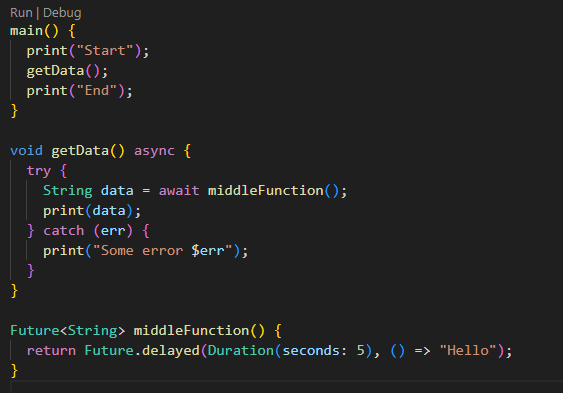
**Important Concept:**

* 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**

1. Create a constant named myAge and initialize it with your age. Write an if statement to print out “Teenager” if your age is between 13 and 19 , and “Not a teenager” if your age is not between 13 and 19.
2. Write a function named youAreWonderful , with a string parameter called name . It should return a string using name , and say something like “You’re wonderful, Bob.”
3. Add another int parameter to above function called numberPeople so that the function returns something like “You’re wonderful, Bob. 10 people think so.”
4. Create a list of names and print all names using list.
5. Create a program that’s reads list of expenses amount using user input and print total.
6. Create an empty list of type string called days. Use the add method to add names of 7 days and print all days.
7. Add your 7 friend names to the list. Use where to find a name that starts with alphabet a.
8. 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.