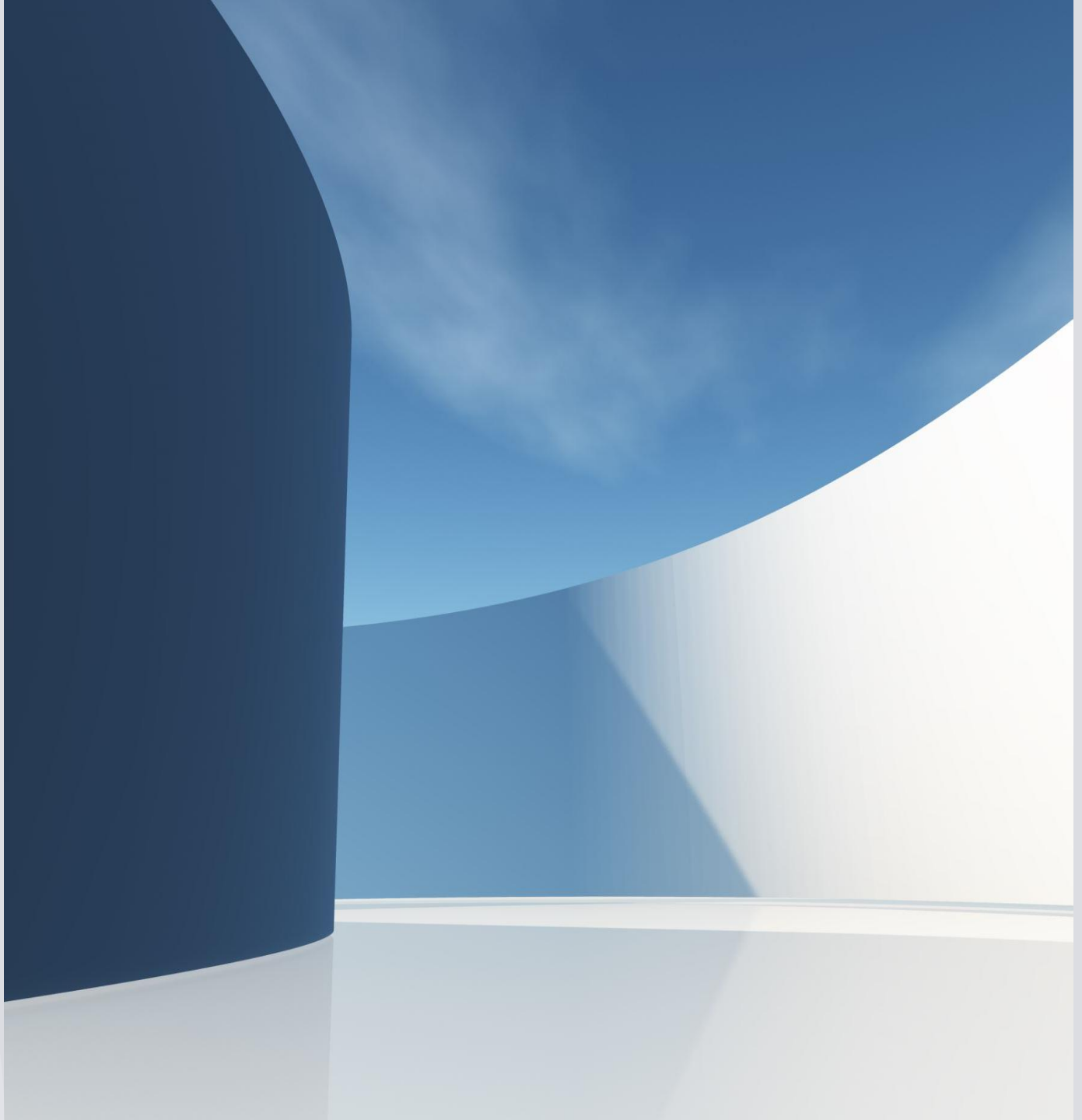
Dart – Data Types



Like other languages **(C, C++, Java)**, whenever a variable is created, each variable has an associated data type. In **Dart language**, there is the type of values that can be represented and manipulated in a programming language.

Data Type	Keyword	Description
Number	num, int, double, BigInt	Numbers in Dart are used to represent numeric literals
Strings	String	Strings represent a sequence of characters
Booleans	bool	It represents Boolean values true and false
Lists	List	It is an ordered group of objects
Maps	Map	It represents a set of values as key-value pairs
Sets	Set	It represents an unordered collection of unique items.

1. **Number:** The number in Dart Programming is the data type that is used to hold the numeric value. Dart numbers can be classified as:

- The **num** type is an inherited data type of the int and double types.
- The **int** data type is used to represent whole numbers.
- The **double** data type is used to represent 64-bit floating-point numbers.

```dart
void main() {
// declare an integer
  int num1 = 2;
// declare a double value
  double num2 = 1.5;
// print the values
  print(num1);
  print(num2);
  var a1 = num.parse("1");
  var b1 = num.parse("2.34");
  var c1 = a1 + b1;
  print("Sum = ${c1}");
}
```

```
2
1.5
Sum = 3.34
```

2. String: It used to represent a sequence of characters. The keyword string is used to represent string literals. String values are embedded in either single or double-quotes.

```
void main() {

  String string = 'Welcome to Dart';
  String str = “Coding is “;
  String str1 = 'Fun’;

  print(string);

  print(str + str1);
}
```

```
Welcome to Dart
Coding is Fun
```

3. Boolean: It represents Boolean values true and false. The keyword bool is used to represent a Boolean literal in DART.

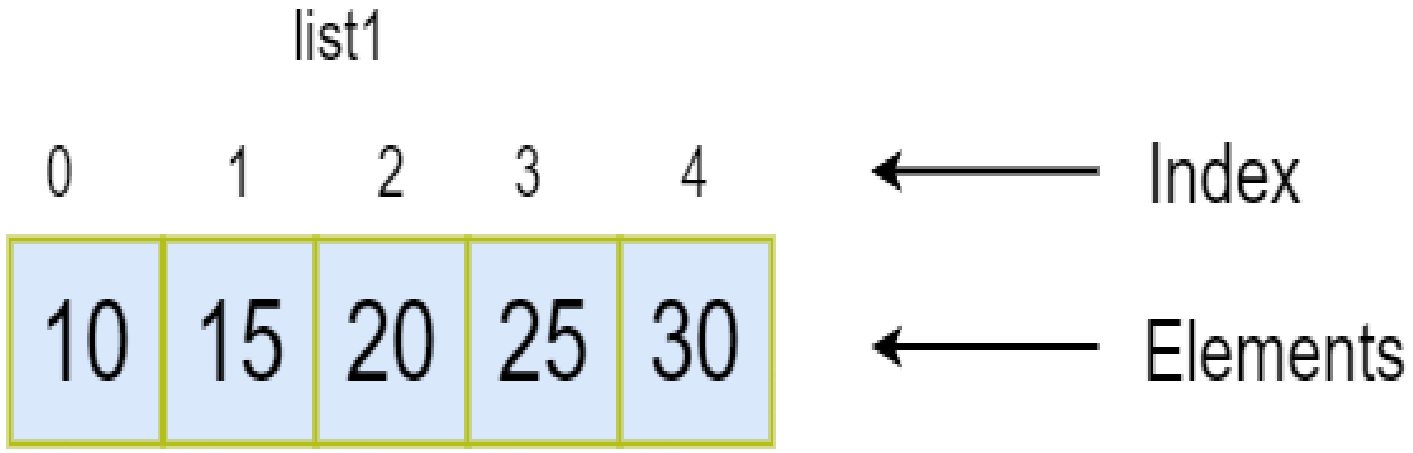
```
void main() {
  String str = 'Coding is ';
  String str1 = 'Fun';

  bool val = (str == str1);
  print(val);
}
```

4. List: Dart List is similar to an array, which is the ordered collection of the objects. The array is the most popular and commonly used collection in any other programming language. The Dart list looks like the JavaScript array literals. The syntax of declaring the list is given below.

```
var list1 = [10, 15, 20,25,30]
```

The Dart list is defined by storing all elements inside the square bracket [ ] and separated by commas (,).

The graphical representation of the list :



Lists Cont

Important information you should know before working with Dart List:

•There are kinds of List: fixed-length list (list's length cannot be changed) & growable list (size can be changed to accommodate new items or remove items)

•Dart List is an ordered collection which maintains the insertion order of the items.

•Dart List allows duplicates and null values.

•While an operation on the list is being performed, modifying the list's length (adding or removing items) will break the operation.

Types of Lists

The Dart list can be categorized into two types -

1. Fixed Length List

The fixed-length lists are defined with the specified length. We cannot change the size at runtime. The syntax is given below.

```
Var list= List.filled(int length,  fill, {bool growable = false})
```

The above syntax is used to create the list of the fixed size. We cannot add or delete an element at runtime. It will throw an exception if any try to modify its size.

The syntax of initializing the fixed-size list element is given below.

```
list_name[index] = value;
```

```dart
void main() {
  var list1 = List.filled(5, 0);
  list1[0] = 10;
  list1[1] = 11;
  list1[2] = 12;
  list1[3] = 13;
  list1[4] = 14;
  print(list1);
}
```

[10, 11, 12, 13, 14]

2. Growable List

The list is declared without specifying size is known as a Growable list. The size of the Growable list can be modified at the runtime. The syntax of the declaring Growable list is given below.

```
// creates a list with values
var list_name = [val1, val2, val3]   Or  // creates a list of the size zero
                                          var list_name = [ ]
```

```
void main() {
  var myList = [10, 20, 30];
  print("Original list: $myList");

  print("Adding elements to the end of the list...");
  myList.add(40);
  myList.add(50);
  print(myList);
}
```

```
Original list: [10, 20, 30]
Adding elements to the end of the list...
[10, 20, 30, 40, 50]
```

List Properties

Below are the properties of the list.

Property	Description
first	It returns the first element case.
isEmpty	It returns true if the list is empty.
isNotEmpty	It returns true if the list has at least one element.
length	It returns the length of the list.
last	It returns the last element of the list.
reversed	It returns a list in reverse order.
Single	It checks if the list has only one element and returns it.

Inserting Element into List

Dart provides four methods which are used to insert the elements into the lists. These methods are given below.

1. The add() Method

This method is used to insert the specified value at the end of the list. It can add one element at a time and returns the modified list object. Let's understand the following example

```
void main() {
  var odd_list = [1, 3, 5, 7, 9];
  print(odd_list);

  odd_list.add(11);
  print(odd_list);
}
```

2. The addAll() Method

This method is used to insert the multiple values to the given list. Each value is separated by the commas and enclosed with a square bracket ([])

```
void main() {
  var odd_list = [1, 3, 5, 7, 9];
  print(odd_list);

  odd_list.addAll([11, 13, 14]);
  print(odd_list);
}
```

```
[1, 3, 5, 7, 9]
[1, 3, 5, 7, 9, 11, 13, 14]
```

3. The insert() Method

The **insert()** method provides the facility to insert an element at specified index position. We can specify the index position for the value to be inserted in the list.

```
void main() {
  List lst = [3, 4, 2, 5];
  print(lst);

  lst.insert(2, 10);
  print(lst);
}
```

```
[3, 4, 2, 5]
[3, 4, 10, 2, 5]
```

4. The insertAll() Method

The insertAll() function is used to insert the multiple value at the specified index position. It accepts index position and list of values as an argument.

```
void main() {
  List lst = [3, 4, 2, 5];
  print(lst);

  lst.insertAll(0, [6, 7, 10, 9]);
  print(lst);
}
```

```
[3, 4, 2, 5]
[6, 7, 10, 9, 3, 4, 2, 5]
```

Removing List Elements

Dart provides following functions to remove the list elements.

- **remove()**

  It removes one element at a time from the given list. It accepts element as an argument. It removes the first occurrence of the specified element in the list if there are multiple same elements. The syntax is given below. list_name.remove(value)

- **removeAt()**

  It removes an element from the specified index position and returns it. The syntax is given below.

  list_name.removeAt(**int** index)

- **removeLast()**

  The removeLast() method is used to remove the last element from the given list. The syntax is given below. list_name.removeLast()

- **removeRange()**

  This method removes the item within the specified range. It accepts two arguments - **start index** and **end index.** It eliminates all element which lies in between the specified range. The syntax is given below. list_name. removeRange();

Dart Iterating List elements

Dart List can be iterated using the **forEach** method.

```
void main() {
  var list1 = ["Smith", "Peter", "Cruise"];
  print("Iterating the List Element");
  list1.forEach((item) {
    print("${list1.indexOf(item)}: $item");
  });

}
```

```
Iterating the List Element
0: Smith
1: Peter
2: Cruise
```

Dart Iterating List elements

Using iterator property to get Iterator that allows iterating.

```
void main() {
  var list1 = ["Smith", "Peter", "Cruise"];
  print("Iterating the List Element");
  var listIterator = list1.iterator;
  while (listIterator.moveNext()) {
    print(listIterator.current);
  }

}
```

```
Iterating the List Element
 Smith
 Peter
 Cruise
```

Dart Iterating List elements

Using every() method

.

```
void main() {
  var list1 = ["Smith", "Peter", "Cruise"];
  print("Iterating the List Element");
  list1.every((item) {
    print(item);
    return true;
  });

}
```

```
Iterating the List Element
 Smith
 Peter
 Cruise
```

Dart Iterating List elements

Using simple for-each loop

```
void main() {
  var list1 = ["Smith", "Peter", "Cruise"];
  print("Iterating the List Element");
  for (var item in list1) {
    print(item);
  }

}
```

```
Iterating the List Element
 Smith
 Peter
 Cruise
```

Dart Iterating List elements

Using for loop with item index

```
void main() {
  var list1 = ["Smith", "Peter", "Cruise"];
  print("Iterating the List Element");
  for (var i = 0; i < list1.length; i++) {
    print(list1[i]);
  }

}
```

```
Iterating the List Element
 Smith
 Peter
 Cruise
```

Combine Lists in Dart

There are 5 ways to combine two or more list:

1. **Using addAll() method to add all the elements of other lists to the existing list**

```dart
void main() {
// Creating lists
  List l1 = ['Welcome', 'to'];
  List l2 = ['Dart'];

// Combining lists
  l1.addAll(l2);

// Printing combined list
  print(l1);
}
```

```
[Welcome, to, Dart]
```

2. Creating a new list by adding two or more list using from and addAll() method of list

```
main() {
// Creating lists
  List l1 = ['Welcome', 'to'];
  List l2 = ['Dart'];

// Combining lists
  var newList = new
List.from(l1)..addAll(l2);

// Printing combined list
  print(newList);
}
```

```
[Welcome, to, Dart]
```

3. Creating a new list by adding two or more list using expand() method of list

We can add all the elements of the list one after another to a new list by the use of expand() method in Dart. This is generally used to add more than two lists together.

```
main() {
// Creating lists
  List l1 = ['Welcome'];
  List l2 = ['to'];
  List l3 = ['Dart'];

// Combining lists
  var newList = [l1, l2, l3].expand((x) => x).toList();

// Printing combined list
  print(newList);
}
```

```
[Welcome, to, Dart]
```

4. Using + operator to combine list

We can also add lists together by the use of **+** operator in Dart. This method was introduced in the **Dart 2.0 update**.

```
main() {
// Creating Lists
  List l1 = ['Welcome'];
  List l2 = ['to'];
  List l3 = ['Dart'];

// Combining Lists
  var newList = l1 + l2 + l3;

// Printing combined list
  print(newList);
}
```

```
[Welcome, to, Dart]
```

5. Using spread operator to combine the list

As of **Dart 2.3** update, one can also use the spread operator to combine the list in Dart.

```
main() {
// Creating Lists
  List l1 = ['Welcome'];
  List l2 = ['to'];
  List l3 = ['Dart'];

// Combining Lists
  var newList = [...l1, ...l2, ...l3];

// Printing combined list
  print(newList);
}
```

```
[Welcome, to, Dart]
```

Dart Sets

The Dart Set is the unordered collection of the different values of the same type. It has much functionality, which is the same as an array, but it is unordered. Set doesn't allow storing the duplicate values. The set must contain unique values.

It plays an essential role when we want to store the distinct data of the same type into the single variable. Once we declare the type of the Set, then we can have an only value of the same type. The set cannot keep the order of the elements.

Dart Initializing Set

Dart provides two methods to declare/initialize an empty set. The set can be declared by using the {} **curly braces** proceeded by a type argument, or declare the variable type **Set** with curly braces {}. The syntax of declaring set is given below.

```
var setName = <type>{};
Or
Set<type> setname = {};
```

```dart
void main() {
  print("Initializing the Set");
  Set<String> names = {"James", "Ricky", "Adam"};
  print(names);
}
```

Add Element into Set

Dart provides two methods **add()** and **addAll()** to insert an element into the given set. The **add()** method is used to add the single item into the given set. It can add one at a time when the **addAll()** method is used to add the multiple elements to an existing set.

```
void main() {
  print("Insert element into the Set");
  var names = {"James", "Ricky", "Adam"};
  // Declaring empty set
  var emp = <String>{};
  emp.add("Jonathan");
  print(emp);

  // Adding multiple elements
  emp.addAll(names);
  print(emp);
}
```

```
Insert element into the Set
{Jonathan}
{Jonathan, James, Ricky, Adam}
```

Access the Set Element

Dart provides the **elementAt()** method, which is used to access the item by passing its specified index position. The set indexing starts from the 0 and goes up to size - 1, where size is the number of the element exist in the Set. It will throw an error if we enter the bigger index number than its size.

```
void main() {
  print("Access element from the Set");
  var names = {"James", "Ricky", "Adam"};
  print(names);

  var x = names.elementAt(2);
  print(x);
}
```

```
Access element from the Set
{James, Ricky, Adam}
Adam
```

Dart Finding Element in Set

Dart provides the **contains()** method, which is used to find an element in the set. It accepts the single item as an argument and return the result in Boolean type. If the given element present in the set, it returns true otherwise false.

```
void main() {
  print("Example - Find Element in the given Set");
  var names = <String>{"Peter", "John", "Ricky"};

  if (names.contains("Ricky")) {
    print("Element Found");
  } else {
    print("Element not found");
  }
}
```

```
Example - Find Element in the given Set
Element Found
```

Dart Remove Set Element

The **remove()** method is used to eliminate or remove an element from the given set. It takes the value as an argument; the value is to be removed in the given set.

```
void main() {
  print("Example - Remove Element in the given Set");
  var names = <String>{"Peter", "John", "Ricky"};
  print("Before remove : ${names}");

  names.remove("Peter");
  print("After remove  :  ${names}");
}
```

```
Example - Remove Element in the given Set
Before remove : {Peter, John, Ricky}
After remove  :  {John, Ricky}
```

Dart Remove All Set Element

We can remove entire set element by using the **clear()** methods. It deletes or removes all elements to the given set and returns an empty set.

```
void main() {
  print("Example - Remove All Element to the given Set");
  var names = <String>{"Peter", "John", "Ricky"};

  names.clear();
  print(names);
}
```

```
Example - Remove All Element to the given Set
{}
```

Dart Iterating Over a Set Element

In Dart, the set element can be iterated using the **forEach and for in** methods as following

```
void main() {
   var names = <String>{"Peter", "John", "Ricky"};

  names.forEach((value) {
    print('Value:  $value');
  });
for (var value in names) {
    print('value: $value');
  }

}
```

```
Value:  Peter
Value:  John
Value:  Ricky
```

Dart Set Operations

Dart Set provides the facility to perform following set operations. These operations are given below.

Union - The union is set to combine the value of the two given sets a and b.

Intersection - The intersection of the two set a and b returns all elements, which is common in both sets.

Subtracting - The subtracting of two sets a and b (a-b) is the element of set b is not present in the set a.

```
void main() {
  var x = <int>{10, 11, 12, 13, 14, 15};
  var y = <int>{12, 18, 29, 43};
  var z = <int>{2, 5, 10, 11, 32};
  print("Example - Set Operations");

  print("x union y is -");
  print(x.union(y));

  print("x intersection y is - ");
  print(x.intersection(y));

  print("y difference z is - ");
  print(y.difference(z));
}
```

Example - Set Operations
x union y is -
{10, 11, 12, 13, 14, 15, 18, 29, 43}
x intersection y is -
{12}
y difference z is -
{12, 18, 29, 43}

Dart Set Properties

The few properties of the Dart set as follows.

Properties	Explanations
first	It is used to get the first element in the given set.
isEmpty	If the set does not contain any element, it returns true.
isNotEmpty	If the set contains at least one element, it returns true
length	It returns the length of the given set.
last	It is used to get the last element in the given set.
Single	It is used to check whether a set contains only one element.

Convert Set to List

The Set object can convert into the List Object using the **toList()** method.
The syntax is as follows.

Syntax -

1.List<type> <list_name> = <set_name>. toList();

Note - The type of List must be the same as the type of Set.

```
main() {
// Creating lists
  List l1 = [1, 2, 3, 4, 5];

// Combining lists
  var newSet = l1.toSet();

// Printing combined list
  print(newSet);
}
```

```
{1, 2, 3, 4, 5}
```

Dart Map

Dart Map is an object that stores data in the form of a key-value pair. Each value is associated with its key, and it is used to access its corresponding value. Both keys and values can be any type. In Dart Map, each key must be unique, but the same value can occur multiple times. The Map representation is quite similar to Python Dictionary. The Map can be declared by using curly braces {} ,and each key-value pair is separated by the commas(,). The value of the key can be accessed by using a square bracket [ ].

Declaring a Dart Map

Dart Map can be defined in two methods.

Using Map Literals

To declare a Map using map literal, the key-value pairs are enclosed within the curly braces "{}" and separated by the commas.

```
Map a = {};
var b = {};
Map c = new Map();
var d = new Map();
```

```dart
void main() {
  var student = {'name': 'Tom', 'age': '23'};
  print(student);
}
```

Using Map Constructor

To declare the Dart Map using map constructor can be done in two ways. First, declare a map using **map()** constructor. Second, initialize the map.

```
void main() {
  var student = new Map();
  student['name'] = 'Tom';
  student['age'] = 23;
  student['course'] = 'DataBase';
  student['Branch'] = 'Computer Science';
  print(student);
}
```

{name: Tom, age: 23, course: DataBase, Branch: Computer Science}

Map Methods

The commonly used methods are given below.

- **addAll() -** It adds multiple key-value pairs of other.

```
void main() {
  Map student = {'name': 'Tom', 'age': 23};
  print('Map :${student}');

  student.addAll({'dept': 'Civil', 'email': 'tom@xyz.com'});
  print('Map after adding  key-values :${student}');
}
```

```
Map :{name: Tom, age: 23}
Map after adding  key-values :{name: Tom, age: 23, dept: Civil, email: tom@xyz.com}
```

- **remove() -** It removes the key and its associated value if it exists in the given map.

```
void main() {
  Map student = {'name': 'Tom', 'age': 23};
  print('Map :${student}');

  student.remove('age');
  print('Map after removing given key :${student}');
}
```

```
Map :{name: Tom, age: 23}
Map after removing given key :{name: Tom}
```

- **clear() -** It eliminates all pairs from the map.

```
void main() {
  Map student = {'name': 'Tom', 'age': 23};
  print('Map :${student}');

  student.clear();
  print('Map after removing all key-values :${student}');
}
```

```
Map :{name: Tom, age: 23}
Map after removing all key-values :{}
```

- **forEach() -** It is used to iterate the Map's entries. The syntax is given below.

```
void main() {
  Map student = {'name': 'Tom', 'age': 23};
  print('Map :${student}');
  student.forEach((k, v) => print('${k}: ${v}'));
}
```

```
Map :{name: Tom, age: 23}
name: Tom
age: 23
```

Map Properties

The dart:core:package has Map class which defines following properties.

Properties	Explanation
Keys	It is used to get all keys as an iterable object.
values	It is used to get all values as an iterable object.
Length	It returns the length of the Map object.
isEmpty	If the Map object contains no value, it returns true.
isNotEmpty	If the Map object contains at least one value, it returns true.

1.Numbers:
- int: Represents integer values.
- Example: int age = 25;
- double: Represents floating-point numbers with decimal places.
- Example: double height = 1.75;

2.Strings:
- String: Represents a sequence of characters.
- Example: String name = "John Doe";

3.Booleans:
- bool: Represents a logical value, either true or false.
- Example: bool isStudent = true;

4.Lists:
- List: Represents an ordered collection of objects.
- Example: List<int> numbers = [1, 2, 3, 4, 5];

5.Maps:
- Map: Represents a collection of key-value pairs.
- Example: Map<String, int> studentGrades = {'John': 85, 'Jane': 92};

6.Sets:
- Set: Represents an unordered collection of unique objects.
- Example: Set<int> uniqueNumbers = {1, 2, 3, 4, 5};