

Dart Flash Cards

version: 1.0

by: [Vandad Nahavandipoor](#)

covers: Dart 2.14.2

[Twitter](#), [LinkedIn](#), [YouTube](#)

Data Types

Data types are ways of telling Dart what kind of data stored is stored in the memory. They make it easier for the programmer and for the computer to understand what they are dealing with and at the same time you will minimize the risk of calculating invalid operations such as adding a number to text, an operation whose result might not be easily understood by the computer

More info:

- [Dart - Data Types - GeeksforGeeks](#)
- [Dart Programming - Data Types - TutorialsPoint](#)
- [The Dart type system](#)
- [Dart Data Types - W3Schools](#)
- [Video: Basics of Dart, DataTypes](#)



Examples

```
1 // This is an example of a integer
2 10
3 // this is a string inside single quotation marks
4 'Vandad Nahavandipoor'
5 // this is a double precision value
6 3.14
7 // and this is a list
8 [1, 2, 3]
9 // we can also have hashmaps
10 {'key1': 'value1', 'key2': 'value2'}
```

Variables

A variable is a named piece of information with a data type, that contains data in the memory, be it in the stack or in the heap. Variables usually have a name and a data type and they either contain a value or not (optionality). There are naming conventions for variables and other rules as to what data they can be assigned to based on their data types.

More info:

- [Dart Programming - Variables - TutorialsPoint](#)
- [Dart Variables - W3Schools](#)
- [Variables and types in Dart - Suragch](#)
- [Dart - Variables - GeeksforGeeks](#)
- [Dart Variables - TutorialKart](#)



Examples

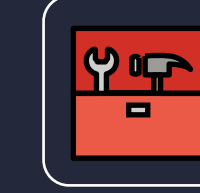
```
1 /*
2 below are all examples of variables whose value can
3 be changed at compile and run time
4 */
5 var foo = 10;
6 foo = foo + 1; // add 1 to foo
7
8 var bar = 'Bar';
9 bar *= 2; // Bar becomes BarBar now
10
11 var baz = 1.2;
12 baz = baz + 3; // baz is now 4.2
```

Dart

Dart is the language that sits behind Flutter, Google's revolutionary UI framework. Quoting Dart's own website: "Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution runtime platform for app frameworks."

Further reading:

- [Dart - Data Types - GeeksforGeeks](#)
- [Dart Programming - Data Types - TutorialsPoint](#)
- [The Dart type system](#)
- [Dart Data Types - W3Schools](#)
- [Video: Basics of Dart, DataTypes](#)



Examples

```
1 /*
2 Dart has many built in features, such as support for
3 many data types such as integers and doubles
4 */
5 final ex1 = 1;
6 final ex2 = 1.2;
7 void func() {
8 // and also support for functions
9 }
10 // and also function type definitions
11 typedef MyCallback = void Function(int);
12 // plus many more
```

Keywords

Keywords are words in a programming language that are reserved for the language itself. In Dart there are variety of keywords that are reserved for the language and shouldn't be used otherwise as variable names for instance. Some of these words are, "break", "false", "true", "finally", "const", "enum", "else", "abstract" and many more which are documented by the good folks at Google.

Further reading:

- [Language tour - keywords - dart.dev](#)
- [Dart Keywords - w3adda](#)
- [Dart Keywords - Javatpoint](#)
- [Dart - Keywords - GeeksforGeeks](#)
- [Keywords \(Reserved Words\) in Dart - Codesansar](#)



Examples

```
1 /*
2 in this case both the words "abstract" and
3 "class" are keywords, or as some may call them,
4 reserved words!
5 */
6 abstract class Animal {
7
8 }
9
10 // here the word "const" is a keyword
11 const name = 'Foo Bar';
12
13 // and here the word "final" is a keyword
14 final age = 32;
```

Double

A double is a double precision floating point value. This is to say that the container of this type can contain values with decimal points, such as 1.2, or 3.14. Integer values on the other hand don't have the capacity to contain a decimal point. Values of type Double can also be "demoted" or type-cast to a value of type integer and vice versa. In the case of Integer to Double, the type will be promoted!

Further reading:

- [Double-precision floating-point format - Wikipedia](#)
- [Double-Precision Floating Point - IBM](#)
- [double class - dartcore library](#)
- [What is a double in Dart? - Educative.io](#)
- [Floating-point arithmetic - Wikipedia](#)



Examples

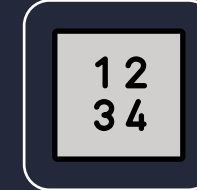
```
1 /*
2 a double value, or a double precision floating
3 point value refers to a value that can contain
4 decimals, in this example, the .2 is the decimal
5 value after the integer value of 1
6 */
7 final fooBar = 1.2;
8 final plusTwo = fooBar + 2;
9 final againPlusTwo = fooBar + 2.0;
10 // a string cannot be added to a double by default
11 final invalid = fooBar + 'Foo Bar'; // compile-error
```

Integer

An integer is a value that can be described by a whole number, up to a limit, dictated by its container variable. An example of an integer is 1,100, 1000, and so on. Integers can be either signed or unsigned. A signed integer can be either positive or negative, while an unsigned integer can only contain positive values, from and including 0 up to the limit dictated by the data type.

Further reading:

- [Integer - Wikipedia](#)
- [Numbers in Dart - Dart dev](#)
- [int class in Dart - api.flutter.dev](#)
- [Integer in Dart - Educative.io](#)
- [Integer class - Pub.dev](#)



Examples

```
1 // a person's age is a good example of an integer
2 final age = 20;
3
4 // you can perform various operators such as
5 // division on an integer
6 final dividedByTwo = age / 2;
7
8 // multiplication is done through the * operator
9 final multipliedByTwo = age * 2;
10
11 // functions or getters can also return
12 // values of type integer
13 final lengthOfName = 'Foo Bar'.length;
```

String

A string in Dart is text usually surrounded by single quotation marks "like so". If the string object itself contains single quotation marks such as "let's", then it's usually surrounded by double quotation marks. Dart recommends that all strings by default be surrounded by single quotation marks unless there is a good reason for them to not to. Strings can be compile time constants.

Further reading:

- [String class - dartcore library - api.dart.dev](#)
- [Dart Programming - String - TutorialsPoint](#)
- [Exploring String methods in Dart - Darshan Kawar](#)
- [Working with Strings in Dart/Flutter - FrontBackend](#)
- [Dart/Flutter String Functions & Operators - Bezkoder](#)



Examples

```
1 // a string is text
2 final myName = 'Vandad Nahavandipoor';
3 /*
4 it can contain numbers as well but this number
5 will not be considered an integer anymore
6 and is from this point on a string, meaning that
7 you cannot add another integer to it
8 */
9 final myAge = '20';
10 // so this code will not compile
11 final invalid = myAge + 30;
```

Class

A class is metadata around a type or multiple types. A class can have properties and methods. Imagine a Person class, a property of this class could be "age" while a method could be "run". Classes can either be abstract or concrete. Classes can then be instantiated in order to create objects. An object is therefore a copy of that class in memory and can be individually modified.

Further reading:

- [Dart Classes and Object - Javatpoint](#)
- [Dart Programming - Classes - TutorialsPoint](#)
- [Language samples - dart.dev](#)
- [Dart - Classes And Objects - GeeksforGeeks](#)
- [Dart Class - TutorialKart](#)



Examples

```
1 /*
2 this is an example of a class with 3 properties namely
3 age, address and name, here the properties are final
4 meaning that their values cannot be changed after being
5 assigned to, the class also has a constructor with
6 the prefix of const
7 */
8 class Person {
9   final String name;
10  final String address;
11  final int age;
12  const Person(this.name, this.address, this.age);
13 }
```

Constants

A constant, or a compile-time constant, is a value whose internals do not change during the entire execution of the program. An example of a constant value in Dart is the value "1", or a constant instance of a class called Person whose first name is "Foo" and last name is "Bar". Those values will not change for the entire lifetime of the program. They can however be changed before compiling the project.

Further reading:

- [Dart final and const - dart.dev](#)
- [Compile-time constants and variables - Newbedev](#)
- [Constants and Variables - Rebus](#)
- [Constant \(computer programming\) - Wikipedia](#)
- [Constants in Programming Language - Toppr](#)



Examples

```
1 /*
2 foo in this example is a compile-time constant
3 meaning that its value cannot be changed after
4 it has been assigned to.
5 */
6 const foo = 'Foo';
7 /*
8 so this code is invalid because
9 foo is a compile-time constant
10 */
11 foo = foo + 'Bar'; // this will not compile
```

Final

Final is a variable modifier that makes a variable assignable only once. Notice that final has nothing to do with constant. A final variable can only be assigned to once, whereas a constant variable can not only be assigned only once, but also has to be a constant! For instance, if you read a user's name from the console into a variable, that value can be stored into a final variable, but not a constant!

Further reading:

- [Final and const - dart.dev](#)
- [Dart - Const And Final Keyword - GeeksforGeeks](#)
- [Difference Between Const and Final in Dart - Jeroen Ouweland](#)
- [Difference between the const and final keywords - Somvarajsinh J](#)
- [Difference Between Constant and Final Explained Example](#)



Examples

```
1 /*
2 this will create an integer variable named
3 "age" whose value is initially set to 20
4 and cannot be changed after it has been declared
5 */
6 final age = 20;
7 final otherAge = age + 20;
8 /*
9 if you try to change the value of this variable
10 after it has been assigned to for the first
11 time, you will receive a compilation error
12 */
13 age = age + 30; // this will not compile
```

Lists

Lists, as their name denote, can contain more than one object at once. Objects are not named. They are accessible by their index. The first item is always at index 0, item 2 is at index 1 and so on. That's why they call lists zero-based, meaning that their indexes start at 0. A list can contain heterogeneous objects, meaning that objects inside the list don't necessarily have to be of the same type.

Further reading:

- [Lists - dart.dev](#)
- [Dart Programming - Lists - TutorialsPoint](#)
- [Dart/Flutter List Tutorial with Examples - Bezkoder](#)
- [List class - dartcore library - Flutter API](#)
- [Dart Lists - W3Schools](#)



Examples

```
1 /*
2 here is an example of 3 strings placed
3 inside a list, the list has 3 items, using
4 the index of 0 you can access 'foo', with the
5 index of 1 you can get to 'bar' and with
6 the index of 2 you can get to 'baz'
7 */
8 const names = ['foo', 'bar', 'baz'];
9 const ages = [20, 30, 44];
10 /*
11 this grabs the values inside both "names" and
12 "ages" and flattens them inside the new list
13 */
14 const all = [...names, ...ages]; // List<Object>
```

Sets

While a list can contain duplicate items, a set's job is to filter out duplicate items. Objects have hash values using which a set determines if two objects are alike. These hash values are usually integers that custom objects can override and define manually. Built-in types have hash values that you don't have to play with in order to have them work together with sets, objects such as strings and integers for instance.

Further reading:

- [Sets - dart.dev](#)
- [Set class - dartcore library](#)
- [Dart Sets - W3Schools](#)
- [Dart Programming - Collection Set - TutorialsPoint](#)
- [Dart Sets - Javatpoint](#)



Examples

```
1 /*
2 a set definition starts with curly brackets and ends
3 with a closing bracket, it cannot contain duplicate values.
4 duplication of values is defined by their hash values.
5 */
6 const ages = {
7   10, 20, 30
8 };
9 // this is not a valid set, and will not compile
10 const names = {
11   'foo',
12   'foo' // this is not allowed
13 }
```

Maps

Maps are key value containers, meaning that for a value to be stored inside the map you need to associate a key to it. The key is then used to retrieve the value. A map or hash map or dictionary as it may be named in other languages is usually used to store structured data. Keys of the map need to be hashable and unique. In Dart you can retrieve a value by key, or just retrieve all the keys or all the values separately.

Further reading:

- [maps - dart.dev](#)
- [Dart Programming - Map - TutorialsPoint](#)
- [Dart Map - zetcode.com](#)
- [Maps in dart - Jay Tillu](#)
- [Dart/Flutter Map, HashMap Tutorial - Bezkoder](#)



Examples

```
1 /*
2 personInfo in this case is a Map<Object, Object>
3 where keys are of type Object and the values
4 are of type Object as well.
5 */
6 final personInfo = {
7   'name': 'Foo bar',
8   'age': 20,
9   'address': 'dummy',
10  10: 20,
11 };
12 // access the name out of personInfo with its key
13 final name = personInfo['name'];
```

Functions

A function is a group of lines of code, or even a single line of code, that has a name, and optionally a return value and arguments. Functions are used for giving contextual meanings to code that are related to each other which perform a specific task. For instance, a Person object might have a function called "run" that performs the running task for that particular person object.

Further reading:

- [functions - dart.dev](#)
- [the main function - dart.dev](#)
- [anonymous functions - dart.dev](#)
- [Dart Programming - Functions - TutorialsPoint](#)
- [Dart function - working with functions in Dart - ZetCode](#)



Examples

```
1 /*
2 here is an example of a person class with an empty
3 function called run, that returns no values (denoted
4 by the void result type) and takes in an argument
5 of type double that denotes the speed in kilometers
6 by which the given person should run when this
7 function is invoked
8 */
9 class Person {
10  void run(double speedInKilometers) {
11    // perform the running task here
12  }
13 }
```

Arguments

Arguments are values that are passed to functions in order to give them more information and context as to how they should perform their tasks. An argument might have a data type and must have a name that the function internally can use in order to refer to that argument. An argument can optionally specify whether it is a required argument or not. Required arguments are prefixed with the "required" keyword.

Further reading:

- [parameters - dart.dev](#)
- [Dart Optional Function Parameters - Zaiste](#)
- [Function Arguments - Flutter by example](#)
- [Dart Optional Parameters - W3Schools](#)
- [Optional Named Parameter - TutorialsPoint](#)



Examples

```
1 /*
2 this function takes in two arguments of type int and
3 it returns the result of adding them together, and the
4 result is of the same data type as the incoming arguments
5 */
6 int add(int value1, int value2) {
7   return value1 + value2;
8 }
9
10 // another example of a function, a short one!
11 int subtract(int value1, int value2) => value1 - value2;
```

Operators

Operators are special functions whose names are symbols. Operators can be prefix, infix or suffix meaning that they can perform operations on data by being specified before that data (prefix), between two pieces of data (infix) or by appearing after an object (suffix). Dart comes with its own built-in operators but you are allowed to define your own operators for existing and new objects.

Further reading:

- [Dart - Data Types - GeeksforGeeks](#)
- [Dart Programming - Data Types - TutorialsPoint](#)
- [The Dart type system](#)
- [Dart Data Types - W3Schools](#)
- [Video: Basics of Dart, DataTypes](#)



Examples

```
1 var foo = 10;
2 /* this is an example of an infix operator that
3 sits between two values */
4 final fooPlusTwo = foo + 2; // fooPlusTwo = 12
5 /* this is a prefix operator that first adds 1
6 to "foo", and then assigns the result to "bar" */
7 final bar = ++foo;
8 /* this is an example of a suffix operator that
9 first assigns the value of foo to baz and then
10 adds 1 to foo */
11 final baz = foo++;
```