Dart: A History

Dart 1.0 was released on November 14th, 2013 by Google and was founded by Lars Bak and Kasper Lund. It aims to help developers build modern web and mobile applications. It covers client, server, and now mobile with Flutter. Coming with a range of tools including a virtual machine, core libraries and package management repository, it lends enough ammunition for you to get started on your next project.

**Dart is Object-Oriented**[#](https://www.educative.io/courses/learn-dart-first-step-to-flutter/g7kmn5r74ok#Dart-is-Object-Oriented)

Smalltalk was released in the 1970s and was one of the first pure object-oriented programming languages. You can now find object-oriented programming everywhere. It has dominated the world of programming languages.

The concept behind object-oriented programming is quite simple: all but the most trivial programs require a particular form of structure.

The most clear-cut way of achieving this particular structure is by using the concept of storage containers. A programming language can be divided into data and operations to be performed on data. We can store specific data and operations in some type of container. Furthermore, these containers are made to be general. Hence, they not only store data and operations but they are themselves values that can be stored in other containers and passed as parameters to other operations. In object-oriented programming, these containers are known as objects.

Alan Kay, the inventor of Smalltalk, remarked that in this way the simplest object has the same construction principle as a full computer: it combines data with operations under a formalized interface.

And now, even though object-oriented programming can be found in a multitude of languages, very few actually follow the principles set by Smalltalk. Dart, on the other hand, is a pure object-oriented programming language with every value being an object.

**Dart: Through the Eyes of JavaScript**[#](https://www.educative.io/courses/learn-dart-first-step-to-flutter/g7kmn5r74ok#Dart:-Through-the-Eyes-of-JavaScript)

Dart is a clean, simple, class-based object-oriented language that has more structure than JavaScript, the programming language it is heavily based on. It’s great for developers that are interested in having a structure in their programming language so that they can easily do refactoring and build large web applications.

According to the founders, one of the things they focused on while creating the language is that it be compatible with the web. Hence, one of the most important components is the Dart to JavaScript compiler. It translates the Dart source code to JavaScript and ensures that you get the same semantics as if you run it on top of the JavaScript virtual machine.

Dart Before You Flutter

In this lesson, we will go over some of the features of Dart that resulted in it being the language of choice for Flutter.

Flutter has been able to grab the attention of the development community by introducing a style that allows for expressive liberty making it a joy to build UIs for mobile apps. It incorporates certain concepts familiar to modern development experiences like reactive programming and widget composition while using the Dart platform as its main base of operations.

The Flutter team evaluated more than a dozen languages and picked Dart because it matched the way they were building user interfaces. Let’s see what Dart offers and why it was chosen by Flutter.

Optimized for UI#

Asynchronous operations let your program complete work while waiting for another operation to finish. Here are some common asynchronous operations:

Fetching data over a network

Writing to a database

Reading data from a file

Most computers, even on mobile platforms, have multi-core CPUs. To take advantage of all those cores, developers traditionally use shared-memory threads running concurrently. However, shared-state concurrency is error-prone and can lead to complicated code. Instead of threads, all Dart code runs inside of isolates. Each isolate has its own memory heap, ensuring that no isolate’s state is accessible from any other isolate.

A programming language optimized for building user interfaces with features for expanding collections, and for customizing the UI for each platform.

Productive Development#

Flutter has a hot reload feature that helps you quickly and easily experiment, build UIs, add features, and fix bugs. Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM). After the VM updates classes with the new versions of fields and functions, the Flutter framework automatically rebuilds the widget tree, allowing you to quickly view the effects of your changes.

Flutter provides static analysis which allows you to find problems before executing a single line of code. It’s a powerful tool used to prevent bugs and ensure that the code conforms to style guidelines.

Fast on all Platforms#

Dart has an AOT (Ahead of Time) compiler, which compiles to fast, predictable, native code that allows almost all of Flutter to be written in Dart. This not only makes Flutter fast but ensures that virtually everything (including all the widgets) can be customized.

Imperative Programming#

The code that appears inside of the main () function gets executed in order of appearance. Let’s modify the code in the snippet above and try printing From Dart along with Hello World.

Void main () {

print ("Hello World");

print ("From Dart");

}

When you run the code above, the print statement on line 2 will execute first and then the second print statement on line 3 will execute.

This style of programming is called imperative programming. It is essentially a programming paradigm wherein you write a set of instructions that execute in sequential order. Imperative programming doesn’t necessarily describe what the program should accomplish, rather it shows how the program should accomplish it. So, in this case, we have created a program that goes first and prints out the text Hello World and then moves on to the next line and prints the text From Dart.

The Hello World program we wrote in a previous lesson was simply displaying some output. But in many cases, we come upon scenarios where we need to take input from the user and print that input onto the console.

Let’s write a program for a personalized greeting application. The program will ask the user for their name and then display a personalized greeting for that user.

The coding program in this lesson requires input from the user in order for it to execute successfully.

1. Before you press RUN, you must select the >\_STDIN button located next to the RUN button which will provide an input field where you can type your input.
2. After you have typed the input, press RUN to execute the program.

import 'dart:io';

main () {

  print ("please Enter your input information ");

  dynamic x = stdin. readLineSync ();

  print ("Hello " + x);

}

Understanding the Code#

The code above might look a bit intimidating. But don’t worry, let’s go over it one line at a time.

Libraries#

On line 1 of the code above, we are importing the dart:io library.

In computer programming, a library is a collection of similar code that you can reference in your own code. When you import a library, you can access all the code in that library.

Every library has a particular purpose. The dart:io library provides I/O (Input/Output) support for non-web applications. We imported dart:io because the program we wrote requires input from the user.

Taking Input in Dart#

Dart provides multiple methods that can be used to take external input from a user depending on the type of input required by the program. For this course, we will be using the readLineSync() method (function) which reads a line from the input.

The complete expression we will use is stdin.readLineSync(). stdin stands for standard input, letting the compiler know that the program specifically requires some sort of input from the user. readLineSync() is a built-in method which we have already discussed above.

Line 4 is displaying Hello and then concatenating (joining) it with the input provided by the user using the addition (+) operator (a topic we will discuss later in the course).

That’s pretty much all there is to the code written above.

Quiz 1. What constitutes as an application entry point in Dart?

A) Hello World

B) Any named function

C)The main() function

D) All of the above

E) None of the above

Quiz 2. What does it mean for a programming language to be imperative?

A) Describes what a program should accomplish in a sequential manner

B) Describes how a program should accomplish a goal in a sequential manner

C) Describes when a program should accomplish a goal in a sequential manner

D) All of the above

E) None of the above

Quiz 3. What is a library in computer programming?

A) A collection of similar code that you can reference in your own code

B) A collection of text which you can refer to in your own code

C) A collection of all the availaible pre-written code you can reference in your own code

D) All of the above

E) None of the above

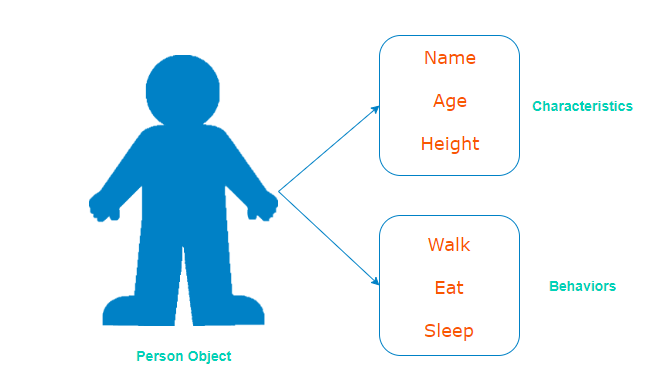
Part II. Data Types and Variables

A Brief Introduction to Objects

Objects in Reality#

Objects are all around us. From the food we eat to the pets we own, everything is an object.

Every object has characteristics and behaviors. For instance, a person has characteristics such as their name, age, and height. A person can also perform behaviors such as walking, eating, and sleeping. These characteristics and behaviors combined define who the person is.



**Objects in Dart#**

In the same way, everything in Dart is an object. Objects in a programming language also have characteristics known as properties and they can also perform behaviors known as methods. Properties represent what the object knows, and methods represent what the object can do.

**Variables**

What Are Variables?

Variables are used for storing information which can then be used by the computer program. Let’s look at this concept from a different angle.

Imagine you’re building a wooden bookshelf and need planks of wood, nails, and different tools to finish your project. To better organize the material, you decide to group similar items together and store them in boxes. This would allow you better access to each item depending on which item you need and when you need it. You have one box with planks, another with nails, and another with tools. However, there’s a problem; you have multiple types of planks, nails, and tools. You have to sort through the right type of box to get the required item. You decide to further divide each box. Multiple boxes can store the same type of item, such as wooden planks, but each box is still unique. There are now two boxes for wooden planks; one with wide planks and another with narrow planks. When you’re done dividing all the material into separate boxes, you end up with an abundance of boxes and can’t seem to find what you’re looking for. Finally, you label each box with a unique name to make each one easier to identify. You now have an organized way to store your material and access it whenever you want.

In the same way, a variable is a small box used to store data. When we assign a value to a variable, we are basically putting something in a box. When you declare a variable, you give it a unique name or identifier, define the type of data it can store, and set its initial value.

Let’s get back to Dart and see how we can declare a variable.

**Declaring a Variable in Dart**

Let’s look at the syntax of how to declare a variable in Dart.

Each spoken language has a general set of rules for how words and sentences should be structured. These rules are collectively known as the language’s syntax. In computer programming languages, syntax serves the same purpose.

Syntax for declaring a variable in Dart

datatype variableName = initial value;

We start off by mentioning the type of data our variable will hold, such as an integer. Then comes the variable’s unique name, followed by an equal sign (=), and the initial value. Don’t forget the semicolon at the very end.

It is a Dart styling convention to name variables using lower camel case. In other words, you should capitalize the first letter of each word except the first word, and use no separators, eg., lowerCamelCase.

Now, let’s map the syntax to the actual code in Dart:

void main()

{

  int num = 5;

}