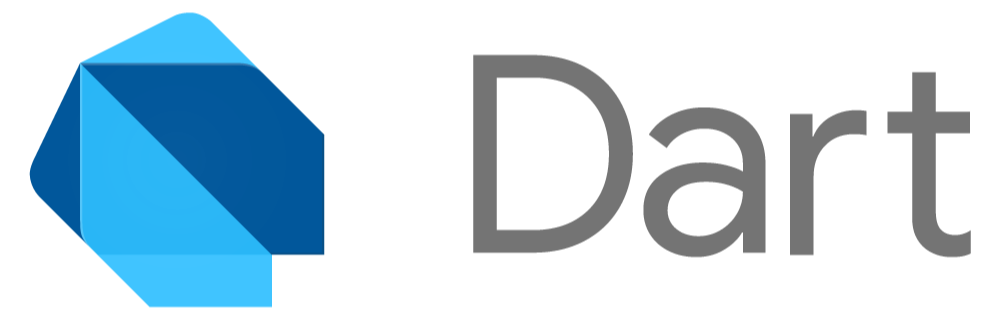
Dart overview



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](https://dart.dev/overview#platform) for app frameworks.

Languages are defined by their *technical envelope*—the choices made during development that shape the capabilities and strengths of a language. Dart is designed for a technical envelope that is particularly suited to client development, prioritizing both development (sub-second stateful hot reload) and high-quality production experiences across a wide variety of compilation targets (web, mobile, and desktop).

Dart also forms the foundation of [Flutter](https://flutter.dev/). Dart provides the language and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code.

## **Dart: The language**

The Dart language is type safe; it uses static type checking to ensure that a variable’s value always matches the variable’s static type. Sometimes, this is referred to as sound typing. Although types are mandatory, type annotations are optional because of type inference. The Dart typing system is also flexible, allowing the use of a dynamic type combined with runtime checks, which can be useful during experimentation or for code that needs to be especially dynamic.

Dart offers [sound null safety](https://dart.dev/null-safety), meaning that values can’t be null unless you say they can be. With sound null safety, Dart can protect you from null exceptions at runtime through static code analysis. Unlike many other null-safe languages, when Dart determines that a variable is non-nullable, that variable is always non-nullable. If you inspect your running code in the debugger, you’ll see that non-nullability is retained at runtime (hence sound null safety).

The following code sample showcases several Dart language features, including libraries, async calls, nullable and non-nullable types, arrow syntax, generators, streams, and getters. To find examples of using additional Dart features, see the [samples page](https://dart.dev/samples). To learn more about the language, take the [Dart language tour](https://dart.dev/guides/language/language-tour).

## **Dart: The libraries**

Dart has [a rich set of core libraries](https://dart.dev/guides/libraries), providing essentials for many everyday programming tasks:

* Built-in types, collections, and other core functionality for every Dart program (dart:core)
* Richer collection types such as queues, linked lists, hashmaps, and binary trees (dart:collection)
* Encoders and decoders for converting between different data representations, including JSON and UTF-8 (dart:convert)
* Mathematical constants and functions, and random number generation (dart:math)
* File, socket, HTTP, and other I/O support for non-web applications (dart:io)
* Support for asynchronous programming, with classes such as Future and Stream (dart:async)
* Lists that efficiently handle fixed-sized data (for example, unsigned 8-byte integers) and SIMD numeric types (dart:typed\_data)
* Foreign function interfaces for interoperability with other code that presents a C-style interface (dart:ffi)
* Concurrent programming using isolates—independent workers that are similar to threads but don’t share memory, communicating only through messages (dart:isolate)
* HTML elements and other resources for web-based applications that need to interact with the browser and the Document Object Model (DOM) (dart:html)

Beyond the core libraries, many APIs are provided through a comprehensive set of packages. The Dart team publishes many useful supplementary packages, such as these:

* [characters](https://pub.dev/packages/characters)
* [intl](https://pub.dev/packages/intl)
* [http](https://pub.dev/packages/http)
* [crypto](https://pub.dev/packages/crypto)
* [markdown](https://pub.dev/packages/markdown)

Additionally, third-party publishers and the broader community publish thousands of packages, with support for features like these:

* [XML](https://pub.dev/packages/xml)
* [Windows integration](https://pub.dev/packages/win32)
* [SQLite](https://pub.dev/packages/sqflite_common)
* [compression](https://pub.dev/packages/archive)

To see a series of working examples featuring the Dart core libraries, take the [library tour](https://dart.dev/guides/libraries/library-tour). To find additional APIs, see the [commonly used packages page](https://dart.dev/guides/libraries/useful-libraries).

## **Dart: The platforms**

Dart’s compiler technology lets you run code in different ways:

* **Native platform**: For apps targeting mobile and desktop devices, Dart includes both a Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler for producing machine code.
* **Web platform**: For apps targeting the web, Dart can compile for development or production purposes. Its web compiler translates Dart into JavaScript.

The [Flutter framework](https://flutter.dev/) is a popular, multi-platform UI toolkit that’s powered by the Dart platform, and that provides tooling and UI libraries to build UI experiences that run on iOS, Android, macOS, Windows, Linux, and the web.

#### **Dart Native (machine code JIT and AOT)**

During development, a fast developer cycle is critical for iteration. The Dart VM offers a just-in-time compiler (JIT) with incremental recompilation (enabling hot reload), live metrics collections (powering [DevTools](https://dart.dev/tools/dart-devtools)), and rich debugging support.

When apps are ready to be deployed to production—whether you’re publishing to an app store or deploying to a production backend—the Dart ahead-of-time (AOT) compiler can compile to native ARM or x64 machine code. Your AOT-compiled app launches with consistent, short startup time.

The AOT-compiled code runs inside an efficient Dart runtime that enforces the sound Dart type system and manages memory using fast object allocation and a [generational garbage collector](https://medium.com/flutter-io/flutter-dont-fear-the-garbage-collector-d69b3ff1ca30).

More information:

* [Get started: Command-line and server apps](https://dart.dev/tutorials/server/get-started)
* [dart tool for running with JIT or AOT compiling to machine code](https://dart.dev/tools/dart-tool)
* [Write command-line apps](https://dart.dev/tutorials/server/cmdline)
* [Write HTTP servers](https://dart.dev/tutorials/server/httpserver)

#### **Dart Web (JavaScript dev & prod)**

Dart Web enables running Dart code on web platforms powered by JavaScript. With Dart Web, you compile Dart code to JavaScript code, which in turn runs in a browser—for example, [V8](https://v8.dev/) inside [Chrome](https://www.google.com/chrome/).

Dart web contains two compiliation modes:

* An incremental development compiler enabling a fast developer cycle
* An optimizing production compiler which compiles Dart code to fast, compact, deployable JavaScript. These effeciencies come from techniques such as dead-code elimination.

More information:

* [Get started: Web apps](https://dart.dev/tutorials/web/get-started)
* [dart compile js](https://dart.dev/tools/dart-compile#js)
* [webdev tool](https://dart.dev/tools/webdev)
* [Web deployment tips](https://dart.dev/web/deployment)

#### **The Dart runtime**

Regardless of which platform you use or how you compile your code, executing the code requires a Dart runtime. This runtime is responsible for the following critical tasks:

* Managing memory: Dart uses a managed memory model, where unused memory is reclaimed by a garbage collector (GC).
* Enforcing the Dart type system: Although most type checks in Dart are static (compile-time), some type checks are dynamic (runtime). For example, the Dart runtime enforces dynamic checks by [type check and cast operators](https://dart.dev/guides/language/language-tour#type-test-operators).
* Managing [isolates](https://dart.dev/guides/language/language-tour#isolates): The Dart runtime controls the main isolate (where code normally runs) and any other isolates that the app creates.

On native platforms, the Dart runtime is automatically included inside self-contained executables, and is part of the Dart VM provided by the [dart run](https://dart.dev/tools/dart-run) command.