Language samples

This collection is not exhaustive—it's just a brief introduction to the language for people who like to learn by example. You might also want to check out the language and library tours, or the Dart cheatsheet codelab.

Language tour

A comprehensive tour, with examples, of the Dart language. Most of the *read more* links in this page point to the language tour.

Library tour

An example-based introduction to the Dart core libraries. See how to use the built-in types, collections, dates and times, streams, and more.

Hello World

Every app has a `main()` function. To display text on the console, you can use the top-level `print()` function:

```
void main() {
  print('Hello, World!');
}
```

Variables

Even in type-safe Dart code, most variables don't need explicit types, thanks to type inference:

```
var name = 'Voyager I';
var year = 1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = {
  'tags': ['saturn'],
  'url': '//path/to/saturn.jpg'
};
```

Read more about variables in Dart, including default values, the `final` and `const` keywords, and static types.

Control flow statements

Dart supports the usual control flow statements:

```
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
}

for (var object in flybyObjects) {
  print(object);
}

for (int month = 1; month <= 12; month++) {
  print(month);
}

while (year < 2016) {
  year += 1;
}
```

Read more about control flow statements in Dart, including `break` and `continue`, `switch` and `case`, and `assert`.

Functions

We recommend specifying the types of each function's arguments and return value:

```
int fibonacci(int n) {
  if (n == 0 || n == 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

var result = fibonacci(20);
```

A shorthand `=>` (*arrow*) syntax is handy for functions that contain a single statement. This syntax is especially useful when passing anonymous functions as arguments:

```
flybyObjects.where((name) => name.contains('turn')).forEach(print);
```

Besides showing an anonymous function (the argument to `where()`), this code shows that you can use a function as an argument: the top-level `print()` function is an argument to `forEach()`.

Read more about functions in Dart, including optional parameters, default parameter values, and lexical scope.

Comments

Dart comments usually start with `//`.

```
// This is a normal, one-line comment.

/// This is a documentation comment, used to document libraries,
/// classes, and their members. Tools like IDEs and dartdoc treat
/// doc comments specially.

/* Comments like these are also supported. */
```

Read more about comments in Dart, including how the documentation tooling works.

Imports

To access APIs defined in other libraries, use `import`.

```
// Importing core libraries
import 'dart:math';

// Importing libraries from external packages
import 'package:test/test.dart';

// Importing files
import 'path/to/my_other_file.dart';
```

Read more about libraries and visibility in Dart, including library prefixes, `show` and `hide`, and lazy loading through the `deferred` keyword.

Classes

Here's an example of a class with three properties, two constructors, and a method. One of the properties can't be set directly, so it's defined using a getter method (instead of a variable).

```dart
class Spacecraft {
  String name;
  DateTime launchDate;

  // Constructor, with syntactic sugar for assignment to members.
  Spacecraft(this.name, this.launchDate) {
    // Initialization code goes here.
  }

  // Named constructor that forwards to the default one.
  Spacecraft.unlaunched(String name) : this(name, null);

  int get launchYear =>
      launchDate?.year; // read-only non-final property

  // Method.
  void describe() {
    print('Spacecraft: $name');
    if (launchDate != null) {
      int years =
          DateTime.now().difference(launchDate).inDays ~/
              365;
      print('Launched: $launchYear ($years years ago)');
    } else {
      print('Unlaunched');
    }
  }
}
```

You might use the `Spacecraft` class like this:

```dart
var voyager = Spacecraft('Voyager I', DateTime(1977, 9, 5));
voyager.describe();

var voyager3 = Spacecraft.unlaunched('Voyager III');
voyager3.describe();
```

Read more about classes in Dart, including initializer lists, optional `new` and `const`, redirecting constructors, `factory` constructors, getters, setters, and much more.

Inheritance

Dart has single inheritance.

```dart
class Orbiter extends Spacecraft {
  num altitude;
  Orbiter(String name, DateTime launchDate, this.altitude)
      : super(name, launchDate);
}
```

Read more about extending classes, the optional `@override` annotation, and more.

Mixins

Mixins are a way of reusing code in multiple class hierarchies. The following class can act as a mixin:

```dart
class Piloted {
  int astronauts = 1;
  void describeCrew() {
    print('Number of astronauts: $astronauts');
  }
}
```

To add a mixin's capabilities to a class, just extend the class with the mixin.

```
class PilotedCraft extends Spacecraft with Piloted {
  // ···
}
```

`PilotedCraft` now has the `astronauts` field as well as the `describeCrew()` method.

Read more about mixins.

Interfaces and abstract classes

Dart has no `interface` keyword. Instead, all classes implicitly define an interface. Therefore, you can implement any class.

```
class MockSpaceship implements Spacecraft {
  // ···
}
```

Read more about implicit interfaces.

You can create an abstract class to be extended (or implemented) by a concrete class. Abstract classes can contain abstract methods (with empty bodies).

```
abstract class Describable {
  void describe();

  void describeWithEmphasis() {
    print('=========');
    describe();
    print('=========');
  }
}
```

Any class extending `Describable` has the `describeWithEmphasis()` method, which calls the extender's implementation of `describe()`.

Read more about abstract classes and methods.

Async

Avoid callback hell and make your code much more readable by using `async` and `await`.

```
const oneSecond = Duration(seconds: 1);
// ···
Future<void> printWithDelay(String message) async {
  await Future.delayed(oneSecond);
  print(message);
}
```

The method above is equivalent to:

```
Future<void> printWithDelay(String message) {
  return Future.delayed(oneSecond).then((_) {
    print(message);
  });
}
```

As the next example shows, `async` and `await` help make asynchronous code easy to read.

```dart
Future<void> createDescriptions(Iterable<String> objects) async {
  for (var object in objects) {
    try {
      var file = File('$object.txt');
      if (await file.exists()) {
        var modified = await file.lastModified();
        print(
            'File for $object already exists. It was modified on $modified.');
        continue;
      }
      await file.create();
      await file.writeAsString('Start describing $object in this file.');
    } on IOException catch (e) {
      print('Cannot create description for $object: $e');
    }
  }
}
```

You can also use `async*`, which gives you a nice, readable way to build streams.

```dart
Stream<String> report(Spacecraft craft, Iterable<String> objects) async* {
  for (var object in objects) {
    await Future.delayed(oneSecond);
    yield '${craft.name} flies by $object';
  }
}
```

Read more about asynchrony support, including `async` functions, `Future`, `Stream`, and the asynchronous loop (`await for`).

Exceptions

To raise an exception, use `throw`:

```dart
if (astronauts == 0) {
  throw StateError('No astronauts.');
}
```

To catch an exception, use a `try` statement with `on` or `catch` (or both):

```dart
try {
  for (var object in flybyObjects) {
    var description = await File('$object.txt').readAsString();
    print(description);
  }
} on IOException catch (e) {
  print('Could not describe object: $e');
} finally {
  flybyObjects.clear();
}
```

Note that the code above is asynchronous; `try` works for both synchronous code and code in an `async` function.

Read more about exceptions, including stack traces, `rethrow`, and the difference between Error and Exception.

Other topics

Many more code samples are in the language tour and the library tour. Also see the Dart API reference, which often contains examples.