Dart

CS 442: Mobile App Development

We'll spend most of today just *talking* about the language, to give you some time to catch up on the documentation and get into the weeds on your own this coming weekend.

Agenda

1. On learning languages
2. Why Dart?
3. Language Overview
4. Tools

On learning programming languages

At this point you should already have learned & mastered two or more languages (e.g., Java & Python). Learning new languages quickly is an important skill!

This is *not* a programming language class --- i.e., I won't be spending much time going over syntax, semantics, etc., outside of working code.

You will need to learn Dart on your own!

How to best learn a language?

– implement data structures & algorithms you are familiar with

– model & solve problems (e.g., Project Euler is a great source)

– be sure to try to incorporate different language features as you encounter them into your "learning" programs

– e.g., I like writing code to generate mazes and solve them

When we get back next week we'll spend a lecture going over examples demonstrating some of the (possibly) less familiar constructs/semantics of the language, and then move onto using it to build Flutter apps.

Why Dart?

We're going to have to learn a new language -- Dart -- before learning how to build apps.

Why can't we just use a language we already know?

How/Why do platform developers decide to adopt new languages when creating platforms?

Introduced as a potential replacement for JavaScript

Or, at least, a serious alternative (in Chrome, to start, where it had its own VM).

Version 1.0 released in 2013 by Google.

Google maintains tons of JavaScript codebases for in-browser applications (e.g., GMail, Google Maps, Google Meet, etc.), so had a good reason to want a more sophisticated language!

But, lots of pushback from web-dev community (why?).

– resistance to new language (boo)
– legacy codebases
– fragmentation of web libraries, communities, etc.

Plans for replacing JavaScript evaporated, but Google continued developing the language.

While it started out mostly as a client-side language, it is increasingly viable for backend development (i.e., "full-stack" work).

Birthed & Evolved with Flutter

Unbeatable synergy!

Flutter was introduced alongside Dart updates that provided improved multi-platform native compilation.

Dart is still a standalone project with compilers, standard library, package management, runtime, etc.

One big selling point is Flutter's "hot reload" feature --- changes to code can be applied to the running application (while being debugged) without restarting and/or losing state. This is enabled by the language execution toolchain (more in a bit).

Familiar syntax + modern features

Important goal: lure developers with immediate productivity!

More on language syntax & features later.

Compiles to native macOS/Windows/Linux executables

Multiple pathways: Just-In-Time (JIT) and Ahead-Of-Time compilation, both of which enable fast execution with different optimizations.

Makes it great for cross-platform development. Ostensibly no loss in efficiency or access to native APIs (really?).

Transpiles to JavaScript

Also makes it suitable for use in browser-based applications.

Flutter apps & libraries are written in Dart

Flutter also makes use of Dart's package management and distribution system.

Note that the Flutter engine, however, is mostly written in C++.

Dart Overview

C-family syntax

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

E.g., C, C++, C#, Java, JavaScript, Rust, etc.

Very familiar. Boring, even. But very easy to get productive in!

Statically typed & Type safe

What is static typing?

Ans: variables have fixed (static) types associated with them --- e.g., via declarations.

```
void main() {
  int age       = 25;
  double height = 5.9;
  String name   = 'Alice';

  print(fortune(name, age));
}

String fortune(String name, int age) {
  // ...
}
```

Why is this useful?

Ans: the compiler can ensure that variables are only ever assigned the correct types, and, by extension, that functions are only ever called with the correct argument types, return the expected result types, etc.

E.g., what would happen if we tried to assign the value in `name` to `age`, or call `fortune` with the wrong types?

Also: latest release (3) supports null safety --- i.e., variables can be declared to be non-nullable. No "null pointer exceptions"!

What is annoying about static typing?

Ans: Having to write type declarations!

Type inference
This means the compiler is (typically) able to *infer* the types of variables based on usage.

```
void main() {
  var age    = 25;
  var height = 5.9;
  var name   = 'Alice';
}
```

Because of the inference engine you can just declare variables using `var` to let the type inference engine do its job (and the Dart style guide prefers this).

Note: you still can't assign `name` to `age`! (Why?)

But, there are times when you might want to provide explicit type declarations (e.g., when you want a variable to use a more general --- e.g., superclass --- type than the specific type of value being assigned to it).

It is also possible to refer to *dynamic* behavior (i.e., where variable types aren't known until runtime), but this is rarely needed and can result in runtime errors.

Fully object-oriented
There are *no primitives* in Dart. Every value is an object (including ints, bools, etc.).

Because of this, *all* variables simply hold references. (Python is also like this.) This means no annoying value vs. reference variable/type dichotomy.

Single-inheritance + Mixins
Avoids pitfalls of multiple inheritance, but with the good.

Generics
What are they? Why do they exist?

```
class Box<T> {
  T _item;

  Box(this._item);

  T getItem() {
    return _item;
  }

  void setItem(T newItem) {
```

```
    _item = newItem;
  }
}

void main() {
  // Note: explicit type declarations aren't needed
  Box<int> int_b = Box(42);
  Box<String> str_b = Box('hello');
}
```

Garbage collected

Expected these days.

Asynchronous support

`async` and `await` are built into the language. Modern solutions to "callback hell".

```
void main() {
  fetchData((data) {
    processA(data, (resultA) {
      processB(resultA, (resultB) {
        processC(resultB, (resultC) {
          print("Final result: $resultC");
        });
      });
    });
  });
}

void fetchData(void Function(String) callback) {
  // Simulating asynchronous data fetching
  Future.delayed(Duration(seconds: 2), () {
    callback("Data");
  });
}

void processA(String data, void Function(String) callback) {
  // Simulating asynchronous processing
  Future.delayed(Duration(seconds: 2), () {
    callback("Processed A");
  });
}
```

```
// Similar processB and processC functions...
```

```
void main() async {
  try {
    final data = await fetchData();
    final resultA = await processA(data);
    final resultB = await processB(resultA);
    final resultC = await processC(resultB);
    print("Final result: $resultC");
  } catch (error) {
    print("An error occurred: $error");
  }
}

Future<String> fetchData() {
  return Future.delayed(Duration(seconds: 2), () => "Data");
}

Future<String> processA(String data) {
  return Future.delayed(Duration(seconds: 2), () => "Processed A");
}

// Similar processB and processC functions...
```

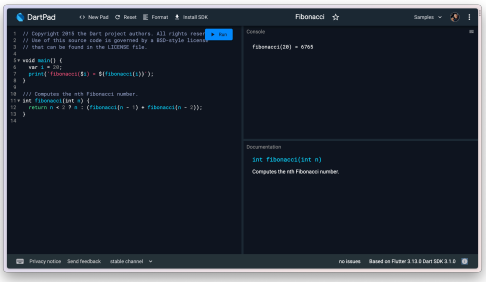
Tools

Lots of different ways to experiment with Dart & Flutter code.

DartPad

https://dartpad.dev/

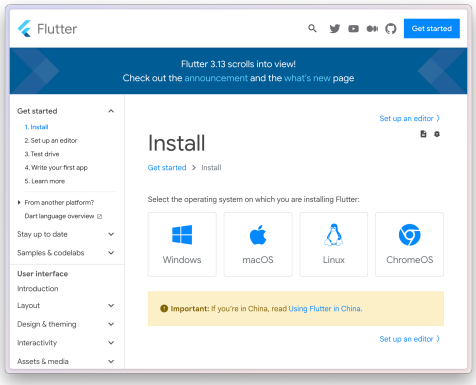
Very handy for playing with code samples from reading, and testing out simple snippets.



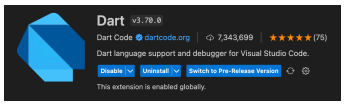
SDK

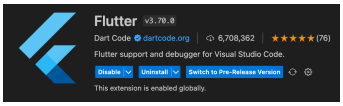
Flutter + Dart

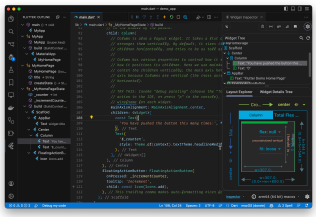
You only need to download and install the Flutter SDK. It bundles the full Dart SDK. Both SDKs include command line tools for compilation, analysis, scaffolding, etc.



Microsoft VSCode



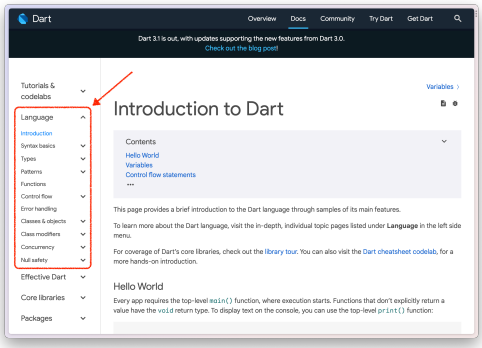




You can use any IDE/text editor you like, but I recommend and will be using Microsoft Visual Studio Code (VSCode).

Instal the Flutter plugin, which also automatically installs the Dart plugin.

For next week



Yes, it's quite a bit, but you should be able to lightly skim through sections that you're familiar with and focus on material that is new/foreign. Reading technical material is another useful skill to practice.

Goal is not to memorize all syntactical constructs and their semantics, but to familiarize yourself enough with them so that when you encounter them in working Flutter code, you aren't completely lost.

If something is truly baffling, write down questions and ask them in class next time! (I'll try to anticipate questions with code demos.)