ESSENTIALS OF DART

Henry Nguyen
Department of Computer Science
University of Arizona
Tucson, AZ 85721
`henryn098@email.arizona.edu`

Adam Cunningham
Department of Computer Science
University of Arizona
Tucson, AZ 85721
`laser@email.arizona.edu`

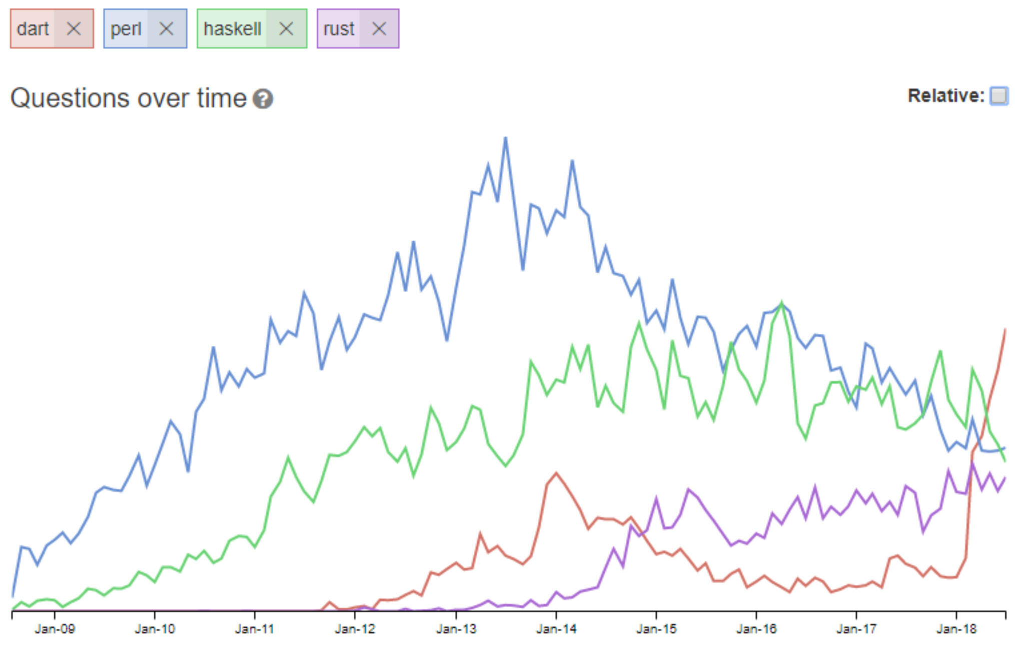
April 14, 2020

ABSTRACT

1 Introduction

2 History

Dart first appeared in 2011 with Elixir and Kotlin. A year later, TypeScript was released by Microsoft. The language was essentially designed to solve the frustrations of JavaScript at the time. Created at Google by Lars Bak and Kasper Lund, Dart is an optionally typed object-oriented language. Dart can act as a superset of JavaScript with the dart2js compiler which included optional static type analysis [1]. Dart is known in the community as the JavaScript killer [2], partly due to the large increase in popularity of Flutter, Google's UI framework for building native interfaces in iOS and Android.



The recent popularity of Dart, coupled with its ongoing support and use in upcoming Google projects makes Dart a compelling language

Flutter is a major project that allows AOT (Ahead of time) and JIT (Just in Time) compilation. This leads to innovative development with hot-reload and a smoother UX [3]. In addition, Flutter is able to render 120 fps on supported devices. Flutter is ran via the Dart platform, and Google is continuing to build and support Dart and Flutter. In 2016, Google began using Dart to create a new operating system, Fuchsia, which is rumored to possibly replace the Android operating system [4].

3 Control Structures

A control structure is a building block in computer programming that allows the direction of a program to change upon given parameters. Languages such as C and Java have identical syntax for the basic control structures in Dart.

3.1 Loops

3.1.1 For loop

A for loop is used to iterate for a fixed number of times, typically for the purposes of counting or traversing a data structure.

```
//do something 10 times
for (int i; i < 10; i = i+1){
    //do something
}
```

3.1.2 While loop

A while loop is used to do something a possibly unknown number of times. The conditional must be a statement which evaluates to true or false. All for loops can be written as while loops. The following calls a function which returns randomly "heads" or "tails". The loop will continue to run until the result is "heads";

```
string heads = "heads";
string result = "";

while (result != "heads") { //executes until result is heads
    result = CoinFlip();
}
```

3.1.3 Do-while loop

Sometimes a user may want the loop to always be entered at least once, then evaluated. This can be accomplished with the do-while syntax

```
do {
    //do something at least once
} while (cond...)
```

3.1.4 For-in loop

A for-in loop is used to iterate over some structure until the end of that structure has been reached. For example, the following code iterates over a list of numbers and sums them together.

```
sum = 0;
for (int i in list){
    sum += i;
}
print(sum);
```

3.1.5 Await-for loop

The await-for loop is used to iterate over a stream of events asynchronously. More can be read about streams in Data Types. "Asynchronous for loop (commonly just called await for) iterates over the events of a stream like the for loop iterates over an Iterable" (dart.dev).

```
Future<int> sumStream(Stream<int> stream) async {
    var sum = 0;
    await for (var value in stream) {
        sum += value;
    }
    return sum;
}
```

3.2 Conditionals

A switch statement is identical to a series of if statements, which takes some value and checks its equality with any number of cases. An error is thrown if a case does not break, but this can be caught, allowing the code to fall through to the next case.

```
switch (val)
case 1:{
    // if val is equal to the integer 1, this portion of the code will execute
    break;
    }
case 2:{
    // if val is equal to 2, this executes.
    break;
}
default:{
    // this always executes if the switch has not been broken out of
}
```

3.2.1 If-else

If-else statements evaluate blocks of code on a specified condition. The else statements are not necessary, and the default implied else is to do nothing.

```
if (cond) {
    print("do something");
} else if (cond){
    print("do something else");
} else{
    print("default");
}
```

3.2.2 Break and Continue

Break and continue statements are used inside loops to alter the normal flow of a program. Break is used to bring the flow of the program outside of the current loop. On the other hand, continue is used to skip the remaining portion of the while loop, and continue back to the beginning, or to a specified label within the loop. Break and continue are similar to goto in other languages. Similar to goto, misuse of break and continue can result in spaghetti code. It is recommended that these be used sparingly if at all. The following would print the numbers 1,2,3...9 (without commas, each on their own line).

```
int i = 0;
while (true){
    i++ //this is equivalent to i = i+1
    if (i < 10){
        print(i);
        continue; //skips the break and restarts the loop
    }
    break; //exits the loop
}
```

3.2.3 Try/Catch

```
try {
    7 ~/ 0
}
on IntegerDivisionByZeroException {
    print('Cannot divide by zero');
}
catch (e) {
    //this is a catch-all block
}
finally {
    print("Dart is cool!");
}
```

3.3 Misc.

A return statement is similar to continue, but it causes the function to be evaluated to the expression after the call to return. The default return value in Dart is null. The following function will evaluate to true.

```
returnTrueFunc(){
    return true;
}
```

4 Data Types

Dart is a object-oriented and statically-typed language and hence, every variable is an object and the type of a variable must be declared by the user. During static analysis, the data type of a variable is inferred from its initial value (section 4.2) [5]. Type argument inference provides the user with an alternative syntax when declaring a variable; helpful towards convoluted variable declarations. Contrary to other languages such as JavaScript, a variable that is declared with the *var* keyword must keep the same data type during reassignment; failing to do so results in a compilation error. On the other hand, the *dynamic* keyword allows a variable to be reassigned to a different data type i.e. type checking is "disabled" during compilation. This mechanism can lead to type safety issues but it is helpful if your data type is *unpredictable* during its lifetime. This results in slower performance because any operation involving that variable must be checked at runtime to ensure that it is legal. Below are the basic data types in Dart.

4.1 Integer

Unlike other languages such as Java where integers are a fixed-size of 32 bits, it varies in Dart. When compiled on the VM, Dart 1 uses "infinite-precision integers (aka bigints)". A BigInt is a built-in JavaScript object that is used for whole numbers larger than $2^{53} - 1$. According to the official Github documentation, "almost every number-operation must check if the result overflowed, and if yes, allocate a next-bigger number type. In practice this means that most numbers are represented as SMIs (Small Integers), a tagged number type, that overflow into "mint"s (medium integers), and finally overflow into arbitrary-size big-ints". This design decision led to users often bit-anding their numbers to "ensure that the compiler can see that a number will never need more than a SMI" e.g. Jenkin's hash function used in Flutter's engine [6]. Upon the release of Dart 2, the size of int was switched to represent a 64 bit two's complement integer [7]. Using dart2js gives a double precision floating point number in the range of $[-2^{53}\ to\ 2^{53}]$ since that is the only number type that JavaScript supports [8].

```
int n = 8;
```

4.2 Doubles

```
double myNum = 8; // inferred as 8.0
```

4.3 Numbers

A number is either an integer or a double.

```
num x = 3;
x = 6.9;
```

4.4 Strings

In programming languages such as C and Java, Strings are an array of characters but in Dart, the character type does not exist and a String is a "sequence of UTF-16 code units". This design allows users to create Strings containing foreign letters, mathematical symbols and emojis. Strings can be initialized with matching single or double quotes and can be multiline. [9].

```
String s1 = "goodbye";
String s2 = '''
This is my multiline string!
'''
```

4.5 Boolean

A boolean is a true or false value. When a data type such as a String is invoked in a boolean expression, boolean conversion occurs. [10]

```
bool val1 = true;
bool val2 = false;

String s1 = "hi";
String s2 = "there";
print(s1 && s2); // false, because boolean conversion turns s1 into false
```

5 Subprograms

6 Summary

References

[1] et al. Dart (programming language). (accessed: 03.26.2020).

[2] David Bolton. The fall and rise of dart, google's 'javascript killer'. (accessed: 03.26.2020).

[3] Wm Leler. Why flutter uses dart. (accessed: 03.26.2020).

[4] et al. Google fuchsia. (accessed: 03.26.2020).

[5] Uday Hiwarale. Dart (dartlang) introduction: Variables and data types. (accessed: 04.13.2020).

[6] Google. engine/hashcodes.dart at master - flutter/engine. (accessed: 04.13.2020).

[7] Google. Dart - fixed-size integers. (accessed: 04.13.2020).

[8] Google. The dart type system | dart. (accessed: 04.13.2020).

[9] Google. String class- dart:core library- dart api. (accessed: 04.13.2020).

[10] Seth Ladd. Booleans in dart. (accessed: 04.13.2020).