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

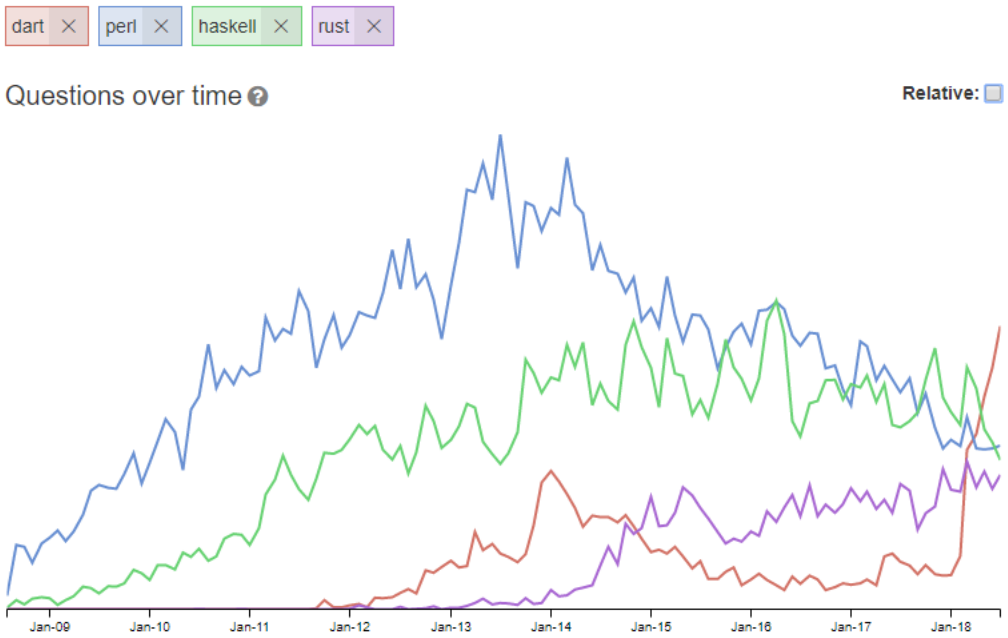
April 7, 2020

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

If you are familiar with control structures in C and Java, then you know the basics of Dart.

3.1 Loops

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
}
```

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();
}
```

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...)
```

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);
```

An 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 like a chain 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
}
```

if-else statements check the conditional. 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");
}
```

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
}
```

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 like 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

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. https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf. (accessed: 03.26.2020).

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