Bilkent University
Fall 2022-2023

CS315
Programming Languages

Homework 3
Subprograms in Dart

Section 2

Deniz Tuna Onguner
22001788

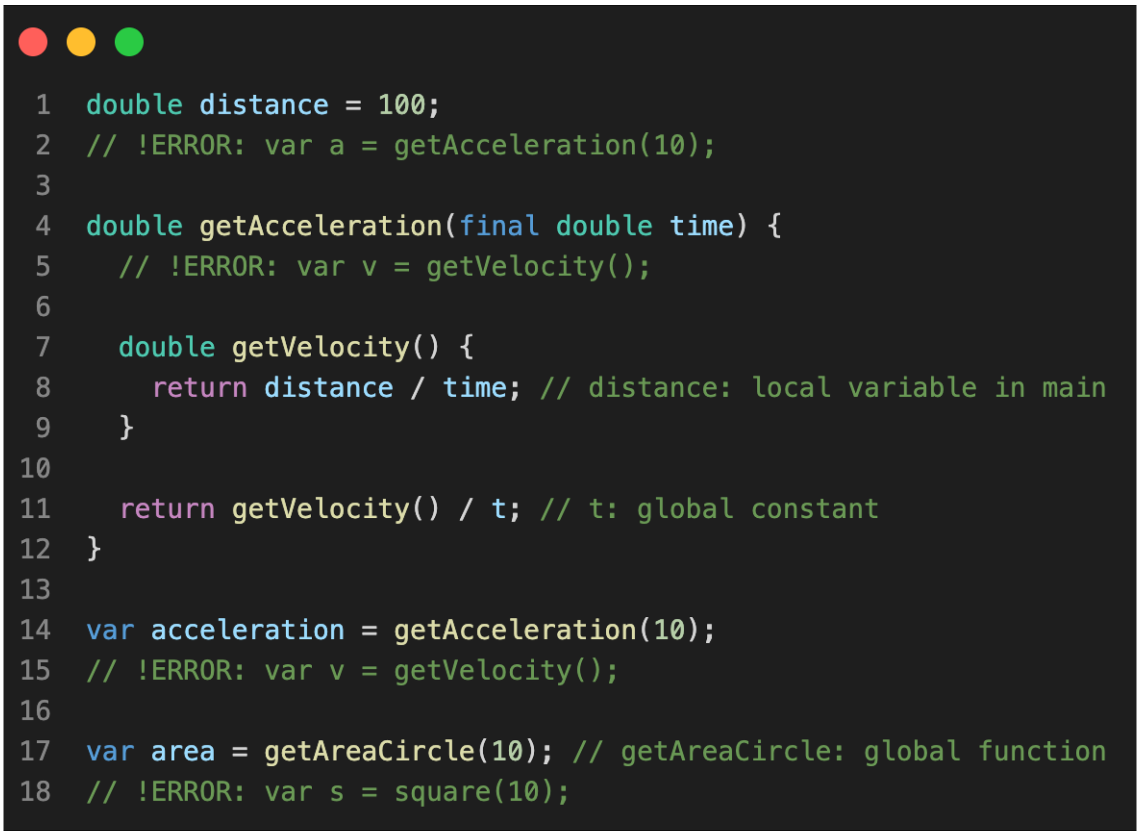
December 17, 2022
Saturday

CONTENTS

1 Subprogram Design Choices and Issues in Dart

1.1 Nested subprogram definitions[1]

In Dart, nested function definitions are allowed, unlike some other well-known programming languages such as Java and Python. Which simply means there might be some function definitions in a scope of another function.



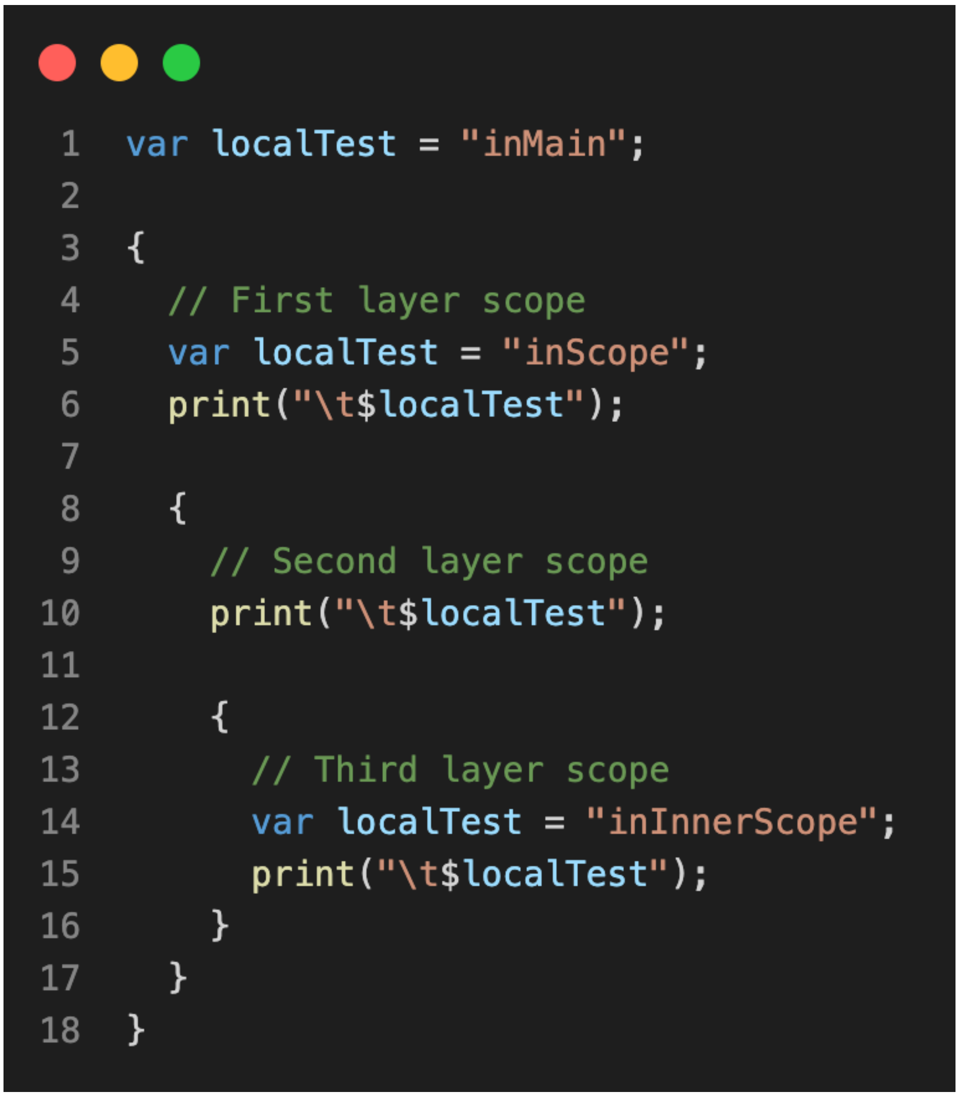
```
 1  double distance = 100;
 2  // !ERROR: var a = getAcceleration(10);
 3
 4  double getAcceleration(final double time) {
 5    // !ERROR: var v = getVelocity();
 6
 7    double getVelocity() {
 8      return distance / time; // distance: local variable in main
 9    }
10
11    return getVelocity() / t; // t: global constant
12  }
13
14  var acceleration = getAcceleration(10);
15  // !ERROR: var v = getVelocity();
16
17  var area = getAreaCircle(10); // getAreaCircle: global function
18  // !ERROR: var s = square(10);
```

In the code segment provided above, a nested function definition block is in the main function. Contrary to globally defined functions, nested definitions must be defined before called. getAreaCircle function on line 17 is defined later parts of the code, outside of main, so it can be called. Because of that design, on line 2, trying to call getAcceleration throws an error. Similarly, nested defined functions cannot be called outside of the scope they are defined in. On line 15, getVelocity cannot be called because of that design. However, nested functions can access higher scope level functions and variables. getVelocity function defined on line 7 is accessing the variable “distance” which is defined in main on line 1. And, getAcceleration is using a global constant “t” on line 11.

[1] “A Tour of the Dart Language Functions,” Dart, accessed December 17, 2022, https://dart.dev/guides/language/language-tour#functions.

1.2 Scope of local variables[2]

Dart is a lexically scoped programming language. Which means that the scope of variables is determined statically, simply by the layout of the code.



```dart
var localTest = "inMain";

{
  // First layer scope
  var localTest = "inScope";
  print("\t$localTest");

  {
    // Second layer scope
    print("\t$localTest");

    {
      // Third layer scope
      var localTest = "inInnerScope";
      print("\t$localTest");
    }
  }
}
```

There is a sample code above containing several scope levels. In each scope, a variable named localTest is defined. And note that there is a localTest in the global scope as well. Each print function is expected to print the localTest which is in the same scope or higher levels of scopes. The first and second ones are going to print “inScope”, and the third one is going to print “inInnerScope”. If there was another print statement for localTest in main, the output would be “inMain”. There is no way to access the global localTest variable in that case from the main function.

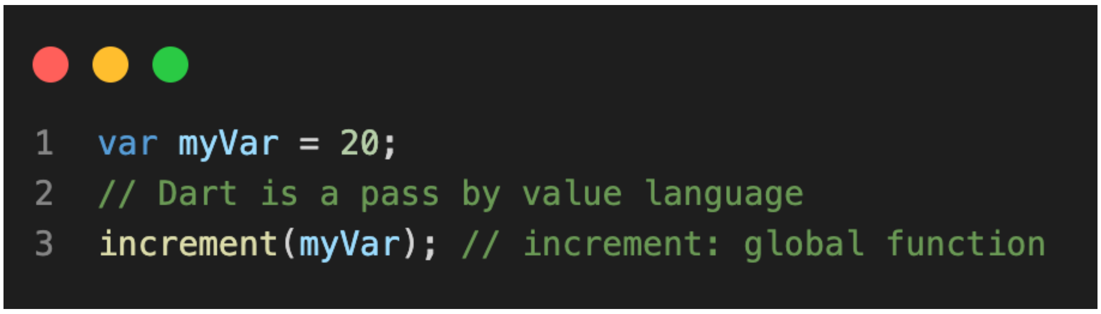
[2] “A Tour of the Dart Language Lexical Scope,” Dart, accessed December 17, 2022, https://dart.dev/guides/language/language-tour#lexical-scope.

1.3 Parameter passing methods[3]

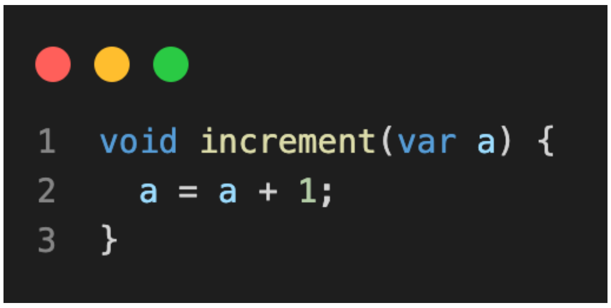
Dart is a pass by value programming language like Java. Contrary to C languages there is no way to pass parameters by reference.

1.3.1 Simple Parameters

Like many other programming languages, in Dart, the parameters can be passed to functions in the following way.



```dart
var myVar = 20;
// Dart is a pass by value language
increment(myVar); // increment: global function
```



```dart
void increment(var a) {
  a = a + 1;
}
```

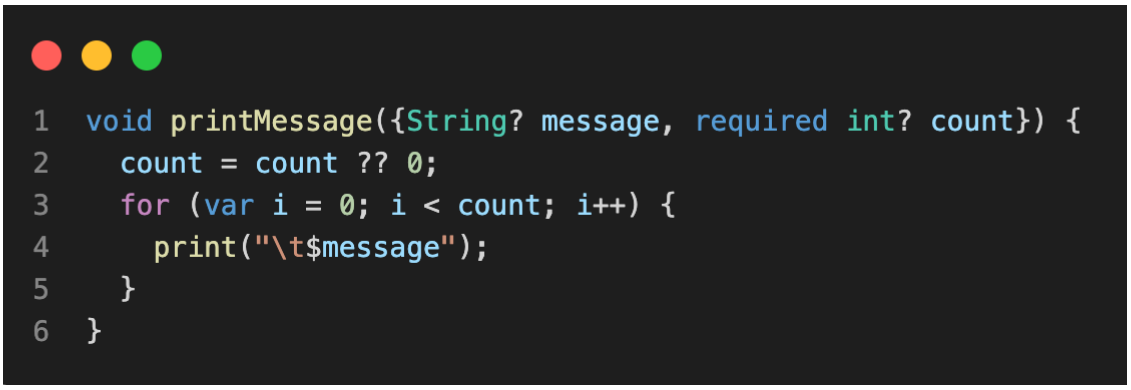
In the code above, the value of variable myVar is not going to alter even though it is passed to a function, since Dart is a pass by value language.

Note that the function increment above, cannot be called without any parameter. The parameter "a" must be provided when function is called.

[3] "A Tour of the Dart Language Parameters." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#parameters.

1.3.2 Named Parameters

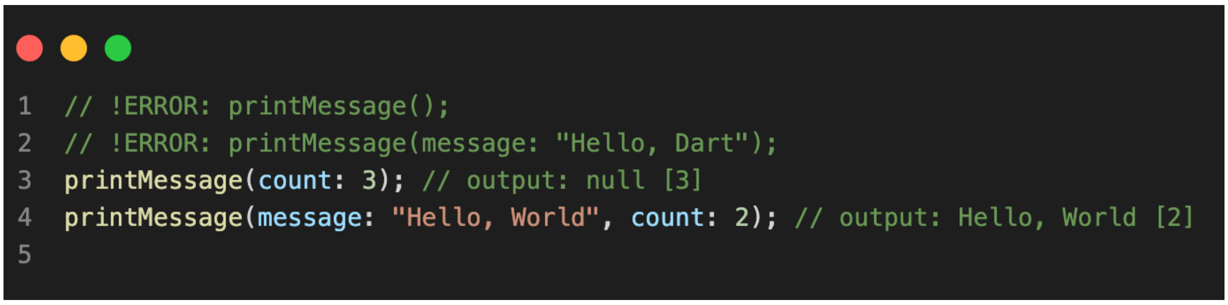
In Dart, parameters can be passed to functions with a name like in



```
1  void printMessage({String? message, required int? count}) {
2    count = count ?? 0;
3    for (var i = 0; i < count; i++) {
4      print("\t$message");
5    }
6  }
```

Swift. By this way, parameters can be passed out of order.

Above, a function printMessage is given, which takes two named parameters “message” and “count”. These two parameters can be null. Parameter count is defined to be required, which means it should be passed a value when function is a called. On the other hand, message is not required to be passed, in such a case the parameter will be null. If count is passed as null, *if null* operator (??) on line 2, will set it



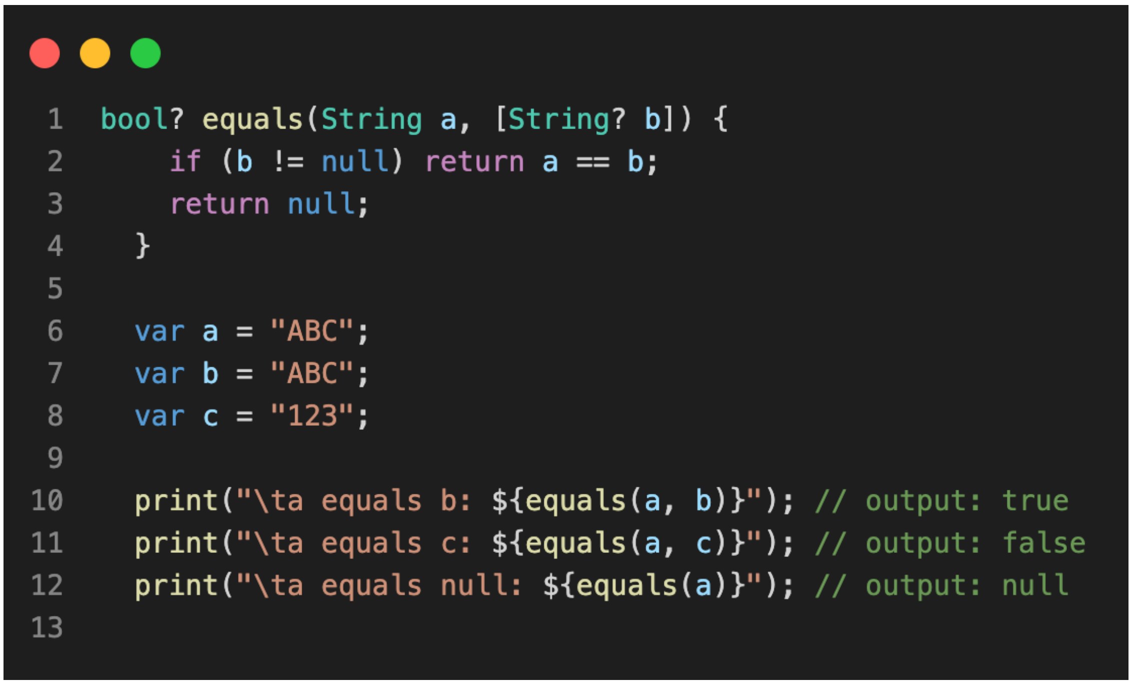
```
1  // !ERROR: printMessage();
2  // !ERROR: printMessage(message: "Hello, Dart");
3  printMessage(count: 3); // output: null [3]
4  printMessage(message: "Hello, World", count: 2); // output: Hello, World [2]
5
```

zero.

As stated, when printMessage is called, parameter count must be passed. So that, line 1 and 2 above throw errors. However parameter message can be null, so line 3 is in a correct syntax. And finally, on line 4, both parameters are passed during call, which is also a correct syntax.

1.3.3 Optional positional parameters

In Dart, functions can take optional parameters, which means these



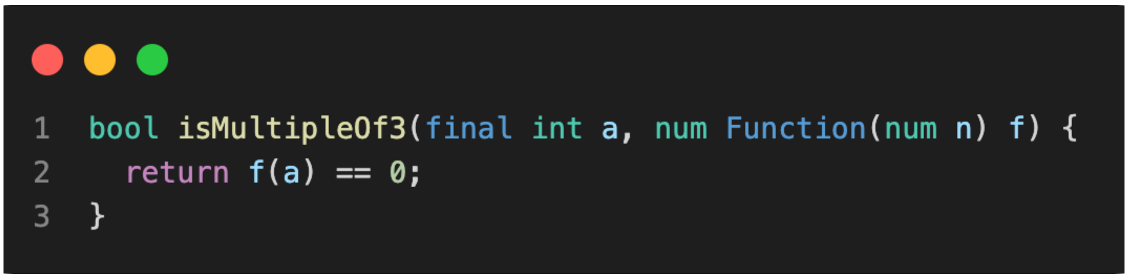
```dart
bool? equals(String a, [String? b]) {
    if (b != null) return a == b;
    return null;
  }

  var a = "ABC";
  var b = "ABC";
  var c = "123";

  print("\ta equals b: ${equals(a, b)}"); // output: true
  print("\ta equals c: ${equals(a, c)}"); // output: false
  print("\ta equals null: ${equals(a)}"); // output: null
```

parameters are not required to be passed when the function is called.

In the code above, a function equals is provided which takes a normal parameter "a" and an optional positional one "b". Since b is an optional parameter, in case it is not passed to the function it will be null. If b is not null, since a must be passed to the function, the function will return whether a equals b.

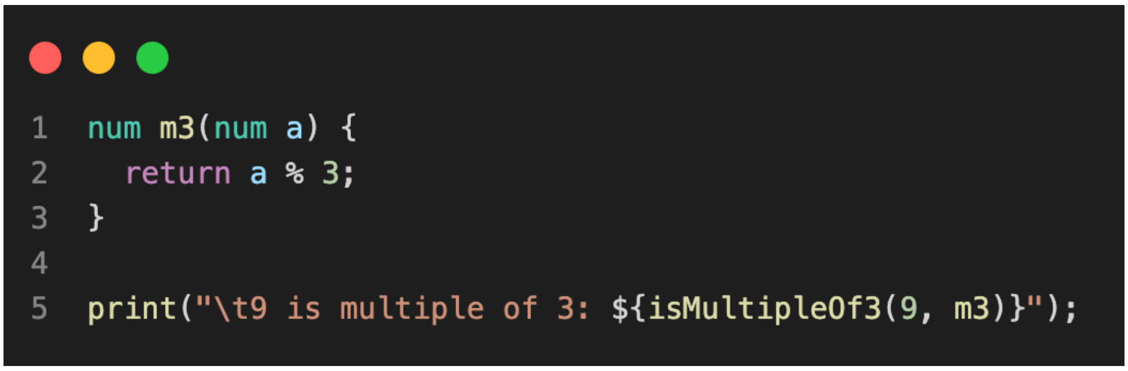
1.3.4 Passing other functions as parameters

In Dart, functions can take other functions as parameters. When passing a function into another one, the details of the parameter function must specify, such as return type and parameters.



```dart
bool isMultipleOf3(final int a, num Function(num n) f) {
  return f(a) == 0;
}
```

In the code block above, a global function isMultipleOf3 is provided, which simply return if the passed integer value is divisible by 3. The function takes two parameters: an integer of “a“ and a function of “f” which return to a numeric value and takes another numeric value “n”.

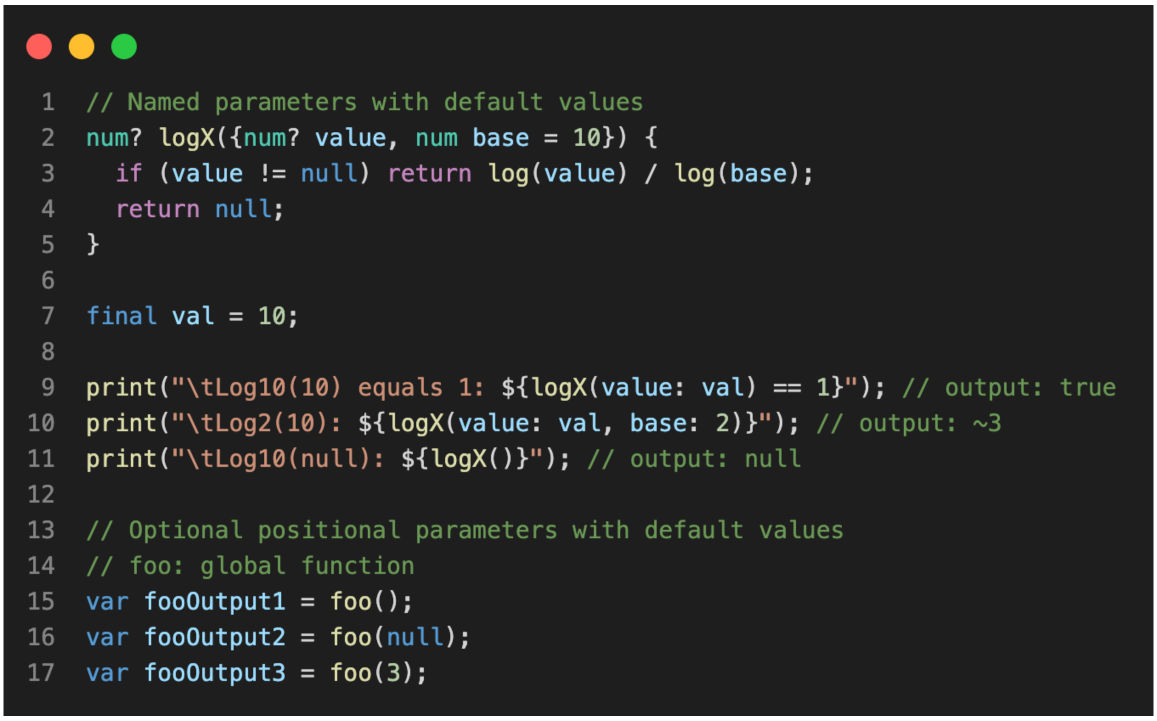


```dart
num m3(num a) {
  return a % 3;
}

print("\t9 is multiple of 3: ${isMultipleOf3(9, m3)}");
```

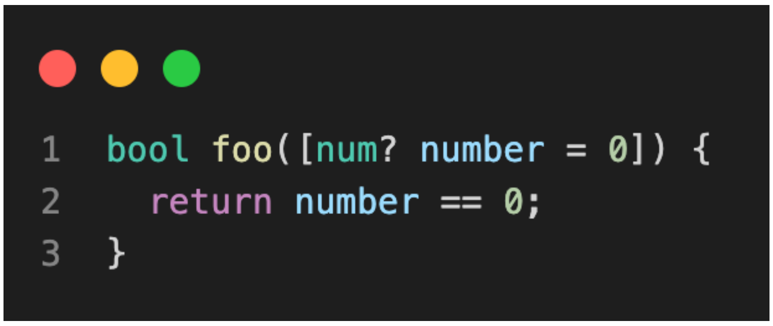
The function isMultipleOf3 is called above. To function parameter, another function of m3 is defined, which return a (mod 3). See that m3 returns to a numeric value and taking another numeric value as parameter, just like the function parameter of isMultipleOf3. If they have not been matched, an error would have been thrown.

1.4 Keyword and default parameters[4]

Dart allows default values for parameters in functions. So that, when a value is not specified by the user, it will be equal to this default value.



```
 1  // Named parameters with default values
 2  num? logX({num? value, num base = 10}) {
 3    if (value != null) return log(value) / log(base);
 4    return null;
 5  }
 6
 7  final val = 10;
 8
 9  print("\tLog10(10) equals 1: ${logX(value: val) == 1}"); // output: true
10  print("\tLog2(10): ${logX(value: val, base: 2)}"); // output: ~3
11  print("\tLog10(null): ${logX()}"); // output: null
12
13  // Optional positional parameters with default values
14  // foo: global function
15  var fooOutput1 = foo();
16  var fooOutput2 = foo(null);
17  var fooOutput3 = foo(3);
```



```
1  bool foo([num? number = 0]) {
2    return number == 0;
3  }
```

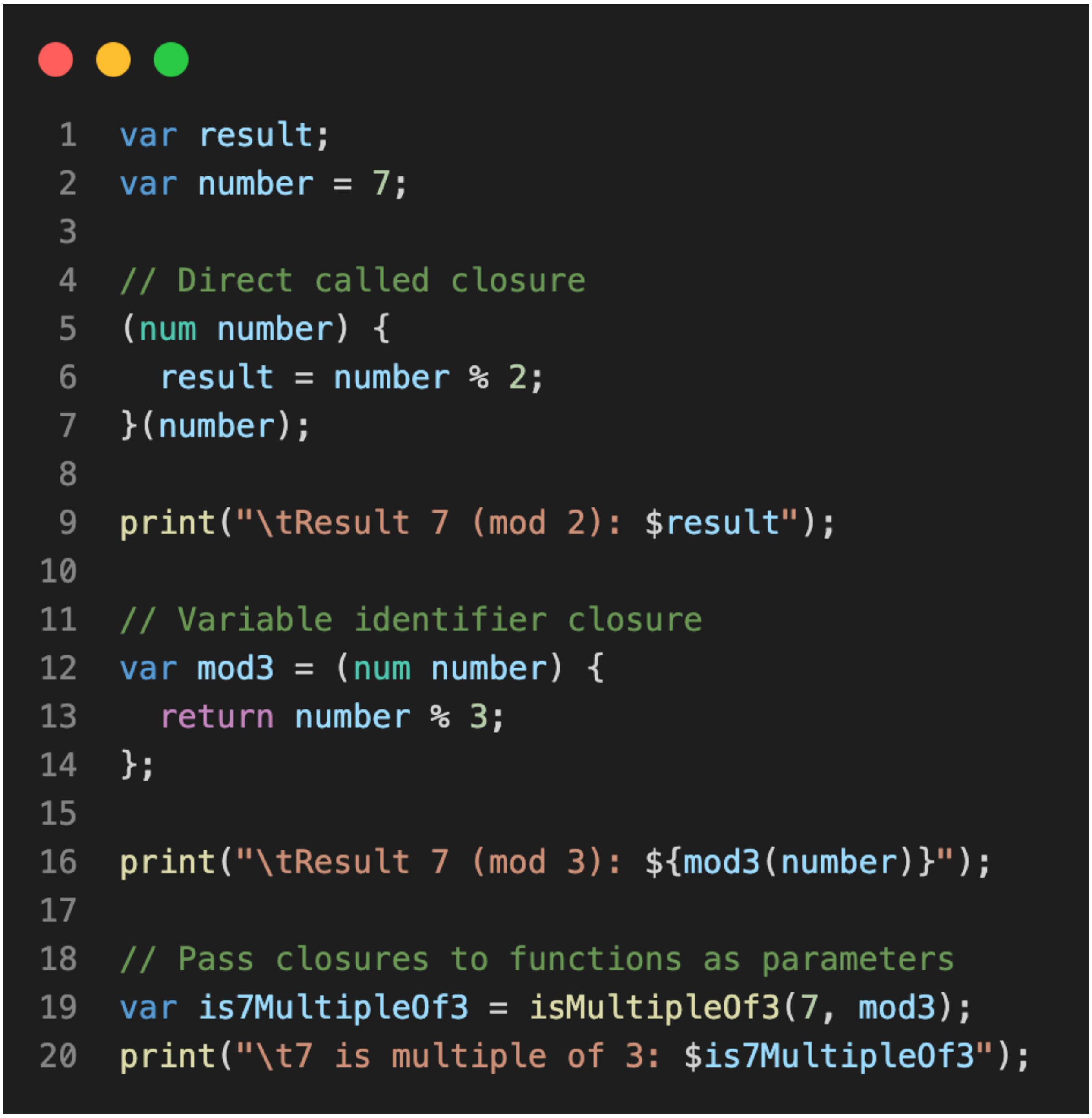
In the code blocks above, some functions are given which are taking parameters with default values. For instance, logX function takes two different parameters: value and base. If parameter base is not passed, it will be equal 10. Since there is a default value is defined for base it does not require null safety, unlike other parameter value. When calling the function, value should be passed, otherwise it will be null, but base can be ignored like on line 9.

Similarly, foo function can also be called without value passing, and parameter number in that case will zero by default. So that, variable fooOutput1 will be true on line 15.

[4] "A Tour of the Dart Language Parameters." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#parameters.

1.5 Closures[5]

Dart supports closures. And, in Dart, closures are a special kind of functions.



```
 1  var result;
 2  var number = 7;
 3
 4  // Direct called closure
 5  (num number) {
 6    result = number % 2;
 7  }(number);
 8
 9  print("\tResult 7 (mod 2): $result");
10
11  // Variable identifier closure
12  var mod3 = (num number) {
13    return number % 3;
14  };
15
16  print("\tResult 7 (mod 3): ${mod3(number)}");
17
18  // Pass closures to functions as parameters
19  var is7MultipleOf3 = isMultipleOf3(7, mod3);
20  print("\t7 is multiple of 3: $is7MultipleOf3");
```

A closure, in Dart, can be defined like a function without an identifier and immediately called after definition. On lines 5 to 7 such an implementation is provided. After execution the variable result will be equal to 1. Or a closure can be kept in a variable to call later on, like on lines 12 to 14. Since functions can be passed as parameters and closures are also functions in Dart, closures can also be passed as parameters like exemplified on line 19 above.

[5] "A Tour of the Dart Language Lexical Closures." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#lexical-closures.

2 Evaluation of Dart in terms of Readability and Writability of Subprogram Syntax

In this report, the programming language Dart is analyzed in terms of its subprograms aka functions. It can be said that, as a comparatively modern programming language, Dart is generally an easy to read and write one.

On the other hand, in terms of nested subprogram definitions, Dart is consider to be hard to read and write since each function might be defined in other functions, it is possible to really long nested definitions. In languages not allowing nested subprograms, like Java, since all functions must be in the global scope, it is easier to differentiate functions from other nested blocks. Moreover, nested functions must be defined earlier on the code before called, which is a kinda low-level design.

Scopes of Dart are easier to read and write. If there are several different variables with the exact same identifiers, the closest one will be called if that identifier is tried to be used. The lower levels of scopes can simply access higher scope levels' variables without requiring any other keywords, unlike Python, which is also a good design choice.

Dart provides several ways for parameter passing, including named parameters and optional positional parameters. Moreover, Dart also supports default values and null safety operators in order to use the exact same function in different ways without overriding it, which makes Dart is very useful. Dart also support closures, which might be very useful for programers time to time.

However, Dart allows functions to take other functions as parameters, which can be considered to be hard to read and write. Also, when the function is defined, the details of the parameter function must also be defined which may not very useful since not every function is passed.

To conclude, as a modern language, Dart has a generally good syntax for subprograms even tough there are some problematic parts are also possible.

3 Personal Learning Strategies and Tools Used

In order to complete this homework, I downloaded Dart compiler into my device, and typed the code on VSCode with Dart extension. The reason I did not apply to online compilers is because I consider them to be relatively slow since they require an active internet connection which I do not have always. To type the code and learn Dart syntax, I read the documentations available on the official webpage of Dart, dart.dev. Analyzed the sample usages and syntax on dart.dev, and created my own examples which I especially designed to answer related questions and issues. Later on, I compared Dart design choices with some other programming languages I am familiar with, such as Java, C++, and Swift to make my evaluation in terms of readability and writability of subprograms.

Compared to the other homework given during the course, this one was slightly easier since only one programming language, Dart, is required to be analyzed and evaluated.

REFERENCES

"A Tour of the Dart Language Functions." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#functions.

"A Tour of the Dart Language Lexical Closures." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#lexical-closures.

"A Tour of the Dart Language Lexical Scope." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#lexical-scope.

"A Tour of the Dart Language Parameters." Dart. Accessed December 17, 2022. https://dart.dev/guides/language/language-tour#parameters.