2022-2023 Fall
CS 315 - Programming Languages

Homework 3 – Report

Subprograms in Dart

Berk Çakar - 22003021 - Section 1

Table of Contents

Subprograms in Dart 3

a) Nested Subprogram Definitions 3

b) Scope of Local Variables 5

c) Parameter Passing Methods 7

d) Keyword and Default Parameters 10

e) Closures 13

Evaluation 15

Learning Strategy 16

References 16

Subprograms in Dart

a) Nested Subprogram Definitions

```dart
void main() {
  print("SUBPROGRAMS IN DART\n");

  print("PART 1: NESTED SUBPROGRAM DEFINITIONS ------------------------------------\n");

  print("SAMPLE 1 ----------------------------------------------------------------\n");

  double circleCircumference(double radius) {
    print("in circleCircumference(radius = $radius)\n");

    // Calculate the circumference for the semi-cirle
    double forSemiCirle(double radius) {
      print("in forSemiCircle(radius = $radius)");
      print("calculating the circumference of the semi-circle of radius $radius");

      double result = pi * radius;

      // Format the output to 2 decimal places
      print(" and the result is ${result.toStringAsFixed(2)}\n");

      return result;
    }

    // Return the full circumference
    return 2 * forSemiCirle(radius);
  }
  // ERROR: The function 'forSemiCircle' is not defined.
  // double half = forSemiCircle(5);

  // Format the output to 2 decimal places
  print("circumference of a circle of radius 5 is: " +
circleCircumference(5).toStringAsFixed(2));

  print("\nSAMPLE 2 ----------------------------------------------------------------\n");
  List makePairs(int a, int b){
    print("in makePairs(a = $a, b = $b)");

    List wrapper() {
      print("  in wrapper()");
      List foo() {
        print("    in foo(), foo() is returning a pair of [a, bar() = b]");
        int bar() {
          print("      in bar() and bar() is returning b = $b");
          return b;
        }

        return [a, bar()];
      }

      return foo();
    }

    return wrapper();
  }
  print("\nList pair of 4 and 6 is: " + makePairs(4, 6).toString());
  // ...
}
```

Figure 1: Examples of nested functions in Dart programming language

```
friberk@Berk-MacBook-Pro  hw3 ❯ dart "cakar_berk.dart"
SUBPROGRAMS IN DART

PART 1: NESTED SUBPROGRAM DEFINITIONS ----------------------------------------

SAMPLE 1 ---------------------------------------------------------------------

in circleCircumference(radius = 5.0)

in forSemiCircle(radius = 5.0)
calculating the circumference of the semi-circle of radius 5.0
 and the result is 15.71

circumference of a circle of radius 5 is: 31.42

SAMPLE 2 ---------------------------------------------------------------------

in makePairs(a = 4, b = 6)
  in wrapper()
    in foo(), foo() is returning a pair of [a, bar() = b]
      in bar() and bar() is returning b = 6

List pair of 4 and 6 is: [4, 6]
```

Figure 2: Output of the code samples given in Figure 1

First of all, in Dart programming language, the term subprogram should be evaluated as functions [1]. In that sense, Dart allows nested functions to be used [2]. As can be understood, nested functions are functions that are defined inside other functions. In Dart, nested functions can access the variables of the outer functions (i.e., the variables available in their static parents) [3]. However, the outer functions cannot access the variables of the nested functions. Similarly, nested definitions are only visible within the scope of the one and only outer function [3]. In other words, only the static parent can call the nested function [3]. For example, in our first sample, the function forSemiCircle() is not visible in the main() but in the circleCircumference().

Regarding nested functions in Dart, detailed code samples can be found in Figure 1. In fact, the code piece as a whole is an example of nested subprogram definitions because all other functions are defined in the main() function. In the first sample, we have a function for calculating the circumference of a circle with the given radius. Inside that function, we define another function that computes the circumference for half of that circle. Overall, we benefit from this nested subprogram definition to complete the calculation. Likewise, the second sample uses nested functions to create a pair (two-element list) of given integers. These samples are important because they demonstrate the concept of nested subprogram definitions in Dart. Outputs of these code samples are also given in Figure 2.

b) Scope of Local Variables

```dart
void main() {
  // ...
  print("\nPART 2: SCOPE OF LOCAL VARIABLES --------------------\n");

  print("SAMPLE 1 ---------------------------------------------\n");

  int g = 8;
  void foo() {
    print("in foo() g = $g, g is in main()");

    int x = 1;
    print("in foo() x = $x, x is in foo()");
    // ERROR: The name 'x' is already defined.
    // int x = 2;

    int t = 7;
    print("in foo() t = $t, t is in foo()");

    void bar() {
      print("  in bar() x = $x, x is in foo()",);
      x = 4; // Update x which is located in foo()
      print("  in bar() x = $x, x in foo() is changed to 4");

      int t = 3;
      print("  in bar() t = $t, t is in bar()");

      int y = 2;
      print("  in bar() y = $y, y is in bar()");
    }
    bar();
    print("in foo() t = $t, t is in foo()");

    // ERROR: Undefined name 'y'.
    // print("in foo() y = $y, y is in bar()");

    print("in foo() x = $x, x is in foo()");
  }
  foo();
  // ERROR: Undefined name 'x'.
  // print("in main() x = $x, x is in foo()");

  print("in main() g = $g, g is in main()");

  // ...
}
```

Figure 3: A code piece that demonstrates the scope of local variables in Dart

```
friberk@Berk-MacBook-Pro  hw3 ❯ dart "cakar_berk.dart"
PART 2: SCOPE OF LOCAL VARIABLES ----------------------------------------------

SAMPLE 1 ----------------------------------------------------------------------

in foo() g = 8, g is in main()
in foo() x = 1, x is in foo()
in foo() t = 7, t is in foo()
  in bar() x = 1, x is in foo()
  in bar() x = 4, x in foo() is changed to 4
  in bar() t = 3, t is in bar()
  in bar() y = 2, y is in bar()
in foo() t = 7, t is in foo()
in foo() x = 4, x is in foo()
in main() g = 8, g is in main()
```

Figure 4: Output of the code samples given in Figure 3

In Dart, local variables are variables that are declared inside a function or a block that defines a scope [1]. Local variables are not accessible outside of the function or the block that defines the scope [4]. However, local variables can be accessed from inner functions or blocks that are defined inside the scope [4]. In other words, Dart uses static (lexical) scoping [1]. Hence, subprograms (functions) can access variables defined in their outer scope [5]. Similarly, outer functions (subprograms) cannot access variables defined in their inner scope [5]. For example, in Figure 3, the function foo() can access the variable g defined in the outer scope (main()). But, the function main() cannot access the variable x defined in the inner scope (foo()). Moreover, it is not possible to define a variable twice in the same scope. On the other hand, it is possible to define a variable with the same name as a variable defined in an outer scope. In this case, the inner variable will hide the outer variable. Lastly, local variables' lifetimes are limited to the scope that defines them. Therefore, local variables are destroyed when the scope that defines them is exited.

In Figure 3 above, the scope of local variables for subprograms in Dart is evaluated in detail. In short, local variables are accessible from inner subprograms. However, local variables are not accessible from outer subprograms. In other words, bar() can access the variables defined in foo() and main(). But, foo() and main() cannot access the variables defined in bar(). Similarly, main() cannot access the variables defined in foo() and bar(). This sample is important and appropriate for this homework because it summarizes the scope of local variables in Dart, especially for functions. Output of the code sample is also given in Figure 4.

c) Parameter Passing Methods

```
class Person {
  String name;
  int age;

  Person(this.name, this.age);

  @override
  String toString() {
    return "Person(name = $name, age = $age)";
  }
}

class Wrapper {
  int first;

  Wrapper(this.first);

  @override
  String toString() {
  return "Wrapper(first = $first)";
  }
}

void main() {

  // ...

  print("\nPART 3: PARAMETER PASSING METHODS ------------------------------\n");

  print("SAMPLE 1: PASS BY VALUE ----------------------------------------\n");

  void variableChanger(int x, String s) {
    print("  executing variableChanger(x = $x, s = $s)");
    x = 2;
    s = "changed";
    print("  in variableChanger, x is changed to $x, s is changed to '$s'");
  }

  int x = 1;
  String s = "unchanged";
  print("in main() x = $x, s = $s");
  variableChanger(x, s);
  print("in main() x = $x, s = $s, x and s are not changed");

  print("\nSAMPLE 1.1: PASS BY VALUE --------------------------------------\n");

  void inverter(bool b) {
    print("  executing inverter(b = $b)");
    b = !b;
    print("  in inverter, b is changed to $b");
  }

  bool b = true;
  print("in main() b = $b");
  inverter(b);
  print("in main() b = $b, b is not changed");

  print("\nSAMPLE 2: PASS BY REFERENCE ------------------------------------\n");
```

```
  void listChanger(List<int> list) {
    print("  executing listChanger(list = $list)");
    list[0] = 2;
    print("  in listChanger, list[0] is changed to ${list[0]}");
  }

  List<int> list = [1, 2, 3];
  print("in main() list = $list");
  listChanger(list);
  print("in main() list = $list, list[0] is changed");

  print("\nSAMPLE 2.1: PASS BY REFERENCE ------------------------------------\n");

  var map = {'a': 1, 'b': 2, 'c': 3};

  void mapChanger(Map<String, int> map) {
    print("  executing mapChanger(map = $map)");

    for (var key in map.keys) {
      map[key] = map[key]! * 2;
    }

    print("  in mapChanger, map is changed to ${map}");
  }

  print("in main() map = $map");
  mapChanger(map);
  print("in main() map = $map, map is changed");

  print("\nSAMPLE 2.2: PASS BY REFERENCE ------------------------------------\n");
  void personChanger(Person person) {
    print("  executing personChanger(person = $person)");
    person.name = "Daniel";
    print("  in personChanger, person.name is changed to ${person.name}");
  }

  Person person = Person("John", 20);
  print("in main() person = $person");
  personChanger(person);
  print("in main() person = $person, person.name is changed");

  print("\nSAMPLE 2.3: PASS BY REFERENCE ------------------------------------\n");

  void increment(Wrapper wrapper) {
    print("  executing increment(wrapper = $wrapper)");
    wrapper.first++;
    print("  in increment, wrapper.first is incremented to ${wrapper.first}");
  }
  int first = 3;
  print("in main() first = $first");
  Wrapper wrapper = Wrapper(first);
  print("in main() first is wrapped in wrapper = $wrapper");
  increment(wrapper);
  print("in main() first is wrapped in wrapper = $wrapper, wrapper.first is incremented by
1");
  // ...
}
```

Figure 5: Examples of parameter passing methods in Dart programming language

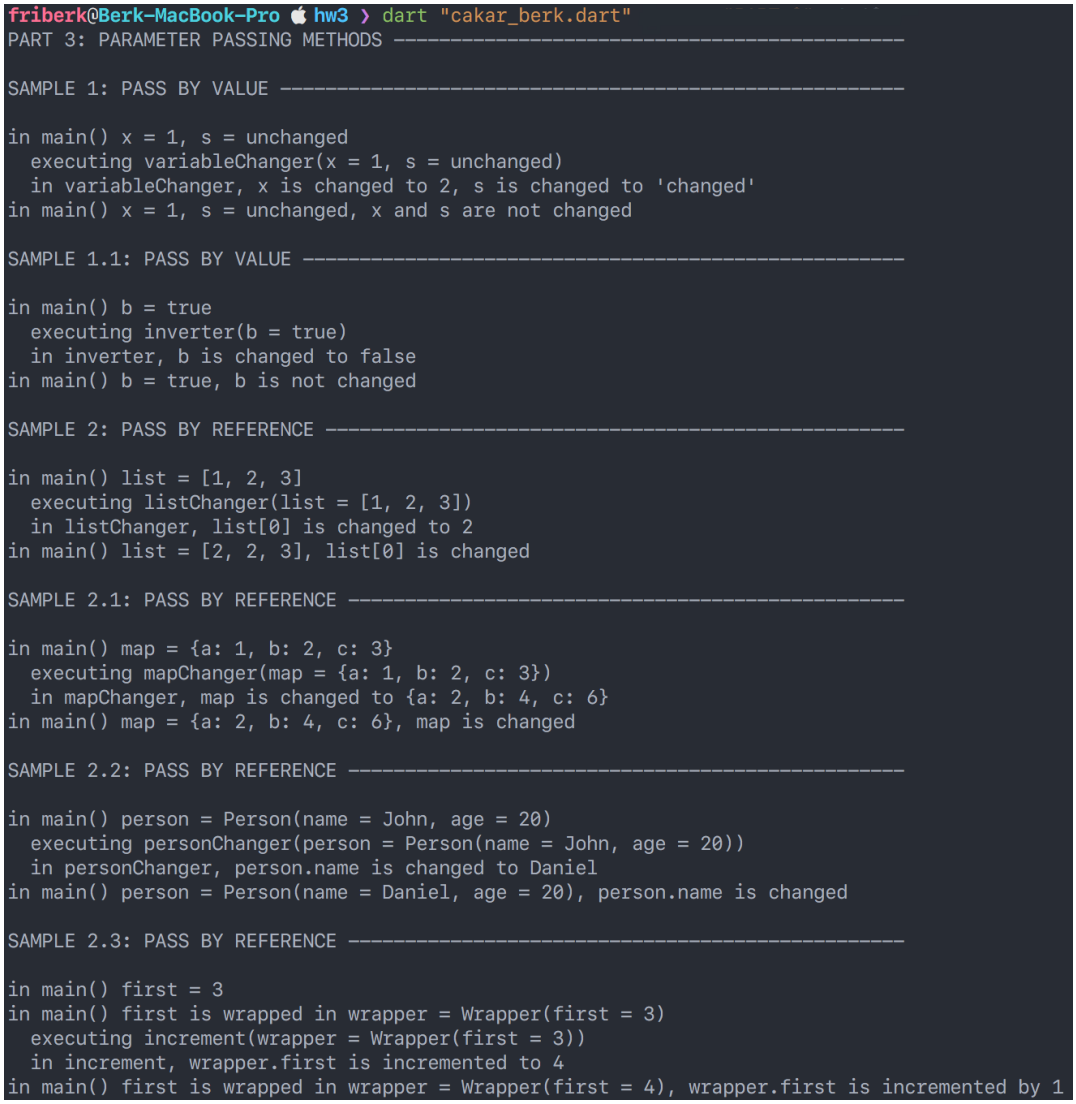

Figure 6: Output of the code samples given in Figure 5

In Dart programming language, parameter passing methods are similar to Java [6]. Objects are passed by reference (including lists and maps); however, all other simpler types are passed by value, such as int, double, string, and boolean [7]. Although the language definition of Dart does not define some types as primitive types, it can be claimed that the types that require relatively less memory space are passed by value, and the types that require relatively more memory space are passed by reference [8]. In short, developers cannot manually decide to pass a parameter by value or by reference, like in C or C++. When a parameter is passed by value, the value of the parameter is copied to the function (to the corresponding formal parameter); therefore, changes made to the formal parameter do not affect the original variable. On the other hand, when a parameter is passed by reference, the function receives a reference to the original variable; hence, the function will reflect any changes made to the original variable.

Regarding the subject, detailed code samples are given in Figure 5. These samples are important and appropriate because they prove which types are passed by value and which are passed by reference in Dart. In short, in these samples, we demonstrated and proved that only objects are passed by reference to the functions in the Dart programming language.

d) Keyword and Default Parameters

```dart
import 'dart:math';

void main() {

  // ...

  print("\nPART 4: KEYWORD AND DEFAULT PARAMETERS ------------------------\n");

  print("SAMPLE 1: KEYWORD (NAMED) PARAMETERS ------------------------------\n");

  int powerPositionalParameters(base, exponent) {
     return pow(base, exponent).toInt();
  }

  int powerNamedParameters({required int base, required int exponent}) {
    return pow(base, exponent).toInt();
  }

  // powerPositionalParameters(2); // ERROR: 2 positional argument expected, but
1 found.
  // powerPositionalParameters(2, 3, 4); // ERROR: Too many positional arguments:
2 expected, but 3 found.
  print("in main() powerPositionalParameters(2, 3) =
${powerPositionalParameters(2, 3)}");
  print("in main() powerPositionalParameters(3, 2) =
${powerPositionalParameters(3, 2)}");

  // powerNamedParameters(base: 2); // ERROR: The named parameter 'exponent' is
required.
  // powerNamedParameters(2, 3); // ERROR: Too many positional arguments: 0
expected, but 2 found.
  print("in main() powerNamedParameters(exponent: 3, base: 2) =
${powerNamedParameters(base: 2,  exponent: 3)}");

  print("\nSAMPLE 1.1: NULLABLE NAMED PARAMETERS ---------------------------\n");

  void printName({String? firstName, required String lastName}) {
    (firstName == null) ? print(lastName) : print('$firstName $lastName');
  }

  printName(firstName: 'Berk', lastName: 'Cakar');
  printName(lastName: 'Cakar');
  // printName(firstName: 'Berk'); // ERROR: The named parameter 'lastName' is
required.

  print("\nSAMPLE 1.2: NAMED AND POSITIONAL PARAMETERS TOGETHER ------------\n");

  /*
    Named and positional parameters can be used together.
    However named parameters must be after positional parameters,
    (i.e., at the end of the parameter list)
  */
  String echo(String greeting, String name, {required String message}) {
    return '$greeting $name. $message.';}
```

```
  print(echo("Hi", "Berk", message: "This is a test message"));
  print(echo("Hi", message: "This is a test message", "Berk"));
  print(echo(message: "This is a test message", "Hi", "Berk"));
  // print(echo("Hi", "Berk", "This is a test message")); // ERROR: Too many
positional arguments: 2 expected, but 3 found.
  // print(echo("Hi", "Berk")); // ERROR: The named parameter 'message' is
required.

  print("\nSAMPLE 2: DEFAULT NAMED PARAMETERS ------------------------------\n");

  double logarithm({required double argument, double base = 10}) {
    return log(argument) / log(base);
  }

  // Or, the following code also works in the same way
  // double logarithm({required double argument, double? base}) {
  //   return (base == null) ? (log(argument) / log(10)) : log(argument) /
log(base);
  // }

  print("in main() logarithm(argument: 100) = ${logarithm(argument: 100)}");
  print("in main() logarithm(base: 2, argument: 64) = ${logarithm(base: 2,
argument: 64)}");

  print("\nSAMPLE 2.1: DEFAULT POSITIONAL PARAMETERS ----------------------\n");

  /*
    Optional positional parameters are declared using square brackets.
    They must be located at the end of the parameter list.
    Lastly, optional positional parameters cannot be used together with named
parameters.
  */
  int integerMultiplication(int first, [int second = 1, int? a]) {
    return first * second;
  }

  // Or, the following code also works in the same way
  // int integerMultiplication(int first, [int? second]) {
  //   return (second == null) ? first : first * second;
  // }

  print("in main() integerMultiplication(2) = ${integerMultiplication(2)}");
  print("in main() integerMultiplication(2, 3) = ${integerMultiplication(2,
3)}");

  // ...

}
```

Figure 7: Examples of keyword and default parameters in Dart programming language

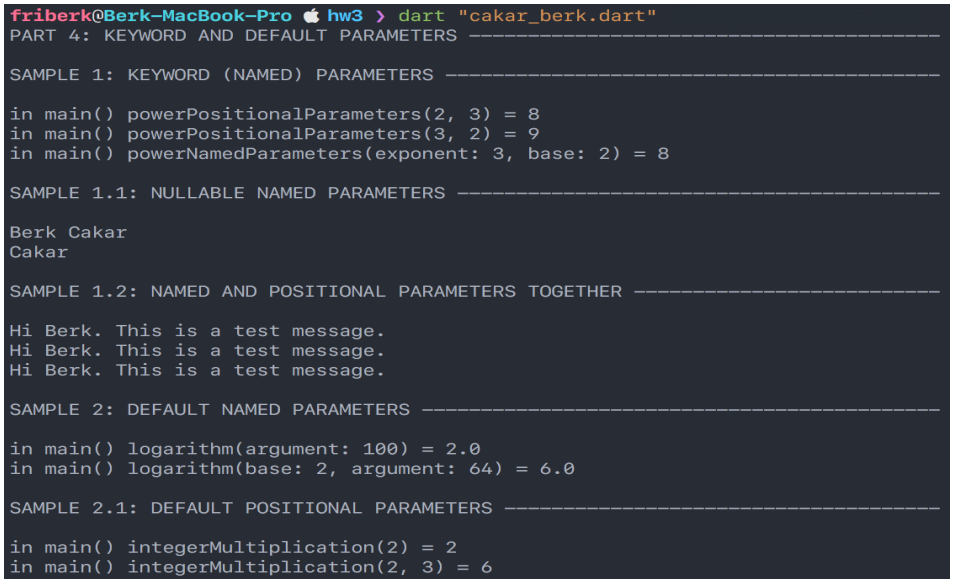

Figure 8: Output of the code samples given in Figure 7

In Dart, both keywords and default parameters are available. Keywords are known as named parameters. By default, all parameters are positional parameters. In order to declare a named parameter, it is necessary to list the parameters in curly braces at the end of the parameter list [9]. Then, inside the curly braces, the declared parameters have to have an indicator of whether they are required or not [9]. In that sense, a named parameter can be either required or optional. For declaring a required named parameter, it is needed to use the keyword 'required' before the parameter name (i.e., required int exponent) [9]. Keywords can be optional in two ways. First, they can be declared as nullable by using the question mark symbol after the parameter type (i.e., int? exponent) [10]. When a keyword is declared as nullable, it will be null in the absence of an argument that corresponds to that keyword [10]. However, declaring a keyword as nullable requires manual null checking in the function body in order to avoid null pointer exceptions [10]. Second, a keyword can be declared with a default value (i.e., int exponent = 1) [10]. According to that, the keyword will be assigned to the default value in the absence of an argument that corresponds to that keyword [10]. In that case, null checking is not required [10]. On the other hand, like optional named parameters, positional parameters can be default parameters as well [9]. For that, these optional positional parameters should be declared at the end of the parameter list in square brackets (i.e., [int exponent = 1, int? a]) [9]. Again, like optional named parameters, optional positional parameters can be either nullable or appear with a default value [9]. Note that optional positional parameters cannot be used together with named parameters [9]. In short, Dart has two types of parameters: positional parameters and named parameters. Named parameters are keywords. And both kinds can have default parameters (i.e., optional). Default parameters can be either nullable or appear with a default value.

About keyword and default values in Dart, detailed code samples are given in Figure 7. These code samples are important and appropriate for this homework because they further investigate the concept of named and optional parameters in the Dart programming language. Explanations of the code segments can be found in the corresponding comment lines and print statements. Outputs of these code samples are also given in Figure 8.

e) Closures

```
void main() {
  // ...
  print("\nPART 5: CLOSURES ------------------------------------------------------------------------\n");


  print("SAMPLE 1: USING CLOSURES THROUGH VARIABLES (ALIASES) ------------------------------\n");

  // Type of closure1 is 'int Function(int, int)'
  var closure1 = (int first, int second) {
    print("  executing closure1(first = $first, second = $second)");
    int sum = first + second;
    print("  in closure1, sum = $sum");
    return sum;
  };

  // or, var closure1 = (int first, int second) => first + second; // using the arrow notation

  print("in main() closure1(7, 5) = ${closure1(7, 5)}");

  print("\nincrement an integer by the given amount using a closure");
  // A closure can return another closure
  var closure2 = (int number) {
    return (int amount) => number + amount;
  };

  int intToBeIncremented = 10;
  print("in main() intToBeIncremented = $intToBeIncremented");
  var intIncrementer = closure2(intToBeIncremented);
  print("in main() intIncrementer(5) = ${intIncrementer(5)}\n");

  // in Dart, closures are objects of the class Function
  // Therefore we can also use 'Function' type instead of 'var'
  double Function(int, double) closure3 = (int first, double second) {
    print("  executing closure3(first = $first, second = $second)");
    double multiplication = first * second;
    print("  in closure3, multiplication = $multiplication");
    return multiplication;
  };

  print("in main() closure3(6, 4.5) = ${closure3(6, 4.5)}");

  print("\nSAMPLE 2: USING CLOSURES THROUGH FUNCTION PARAMETERS ------------------------------\n");

  var closure4 = (int first) => first * 3;

  // A function that takes a closure as a parameter
  void listElemenentManipulator(List<int> list, Function(int) operation) {
    for (int i = 0; i < list.length; i++) {
      list[i] = operation(list[i]);
    }
  }

  List<int> integerList = [1, 2, 3, 4, 5];
  print("in main() list = $integerList");
  listElemenentManipulator(integerList, closure4);
  print("after listElemenentManipulator(integerList, closure4)");
  print("in main() list = $integerList");

  // ...
}
```

Figure 9: Examples of closures in Dart programming language

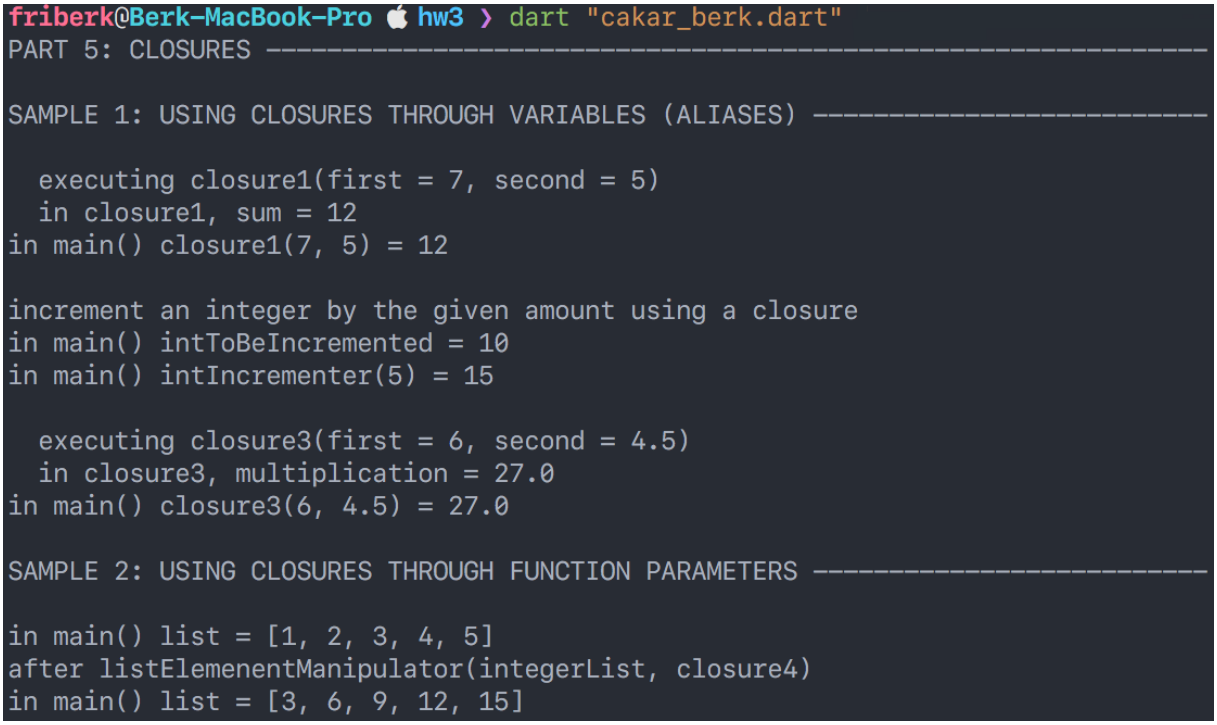

Figure 10: Output of the code samples given in Figure 9

In general, a closure is a bundle of a subprogram that stores a function with its lexical (static) environment [11]. In Dart programming language closures are supported. Interestingly, in Dart, functions are instances of Functions class, therefore we can assign functions to variables and obtain closures [12]. In particular, In Dart, closures can be declared in two ways: (type argument1, type argument2, type argument3, ..., type argumentN) { // statements } and (type argument1, type argument2, type argument3, ..., type argumentN) => expression [12]. In that sense, if a closure contains a single line statement, it can be declared with the arrow notation (i.e., "(int first, int second) => first + second" forms a closure) [12]. Regarding closures in Dart, detailed code samples can be found in Figure 9. For example, in the second sample, it is showed that closures can appear as a parameter in a function. These code samples are important and appropriate for this homework because they summarize the concept of closure in Dart. Outputs of these code samples are also given in Figure 10.

Evaluation

After understanding the subprogram syntax in Dart, I think Dart has a very good level of readability and writability. First, the general syntax is very similar to C-family programming languages such as C, C++, Java, and more. This aspect makes reading and writing code in Dart easy for those familiar with C-family programming languages. Coming back to design issues for this homework, first of all, nested functions are powerful and well-designed in Dart. Because with nested functions, it is possible to separate, divide, or abstract the ongoing logic in a function into smaller nested subprograms. From my perspective, this ease of abstraction is very important for the readability of the code. Also, it increases the writability and reusability of the code since it is possible to write the code more modularly. Secondly, in terms of the scope of local variables, Dart follows a conventional design we already know from C-family programming languages. Namely, Dart uses static scoping which is straightforward to understand. Additionally, Dart supports the redefinition of a variable in the inner scope, which is also an excellent design decision (Java does not support that). Therefore, scoping rules in Dart increase the readability and writability of the language. However, Dart could be more powerful in terms of parameter passing. Dart has a similar logic in passing parameters to functions such as Java and Python. According to that, data types such as int, double, and String are passed by value, whereas objects, lists, and maps are passed by reference. I understand that this design decision is made for the sake of performance and reliability. However, I think it would be more powerful if Dart supported passing parameters by value or reference according to the developer's decision, like using pointers in C language. This would increase the flexibility of the language and make it more powerful. Nevertheless, it can be said that parameter passing does not directly affect the readability and writability of the language. For keyword and default parameters, Dart has an outstanding design. First of all, Dart supports keyword (named) parameters which is a very good design decision. With this feature, parameters can be passed to functions in any order. This is very useful when the number of parameters is large; thus, it increases the readability and writability. Also, it is possible to set keywords as optional and make them default if necessary. Moreover, since this required/optional setting is explicit, reading and understanding the parameter definitions is very easy. Furthermore, Dart explicitly requires the keywords and optional parameters to appear at the end of the parameter list, which is, again, a good design decision. This aspect increases the writability since it removes the confusion of where to put the keywords and default parameters, like in Python. Lastly, I think the language designers have done a very good job for closures in Dart. Unlike in Python and Javascript, the syntax and logic for closures are easy to understand. This simplicity increases the readability and writability of the language. Overall, in terms of the readability and writability of subprogram syntax, Dart is one of the best programming languages I have encountered. And considering the previous two homework, I think Dart is a very good programming language which does a good job on almost every single programming language evaluation criterion.

Learning Strategy

While completing this assignment, I generally tried to refer to the official Dart programming language documentation. But unfortunately, the explanations here cover the very simplistic versions of the concepts asked in the assignment prompt, so I only got a little use from the official documentation in this homework. Therefore, since I could find much comprehensive information from the official documentation, I performed detailed searches over the internet. For example, for the first design issue, I encountered with the blog of a software developer; hence, I benefited from the information there. Similarly, for the second design issue, I have read the articles on Tutorialspoint and GeeksForGeeks, which are credible platforms. However, for the third design issue, interestingly, my main source was a conversation on the official mailing list of the Dart programming language. Lastly, for the fourth and fifth design issues, I mainly referred to the personal blogs of software developers to gather information. Overall, in addition to the insufficient official documentation, I also referred to reliable tech sources on the internet, including Q&A websites such as StackOverflow. Since I already had the necessary Dart compiler installed on my computer, I did not use any online compilers. However, I still ran my codes on the Dijkstra server to ensure that there were no platform-related issues.

Also, in addition to the information I gathered from the internet, I performed the experiments on the given design issues regarding the subprograms in Dart by using the trial-error methodology. Therefore, I was able to explore the concepts better. In fact, I benefited from this methodology a lot in the third and fourth design issues.

References

[1] https://dart.dev/guides/language/language-tour

[2] https://docs.bongthorn.dev/pl/learn-dart-first-step-to-flutter/exploring-funtions/nested-functions

[3] https://riptutorial.com/dart/example/10666/function-scoping

[4] https://www.tutorialspoint.com/lexical-scoping-in-dart-programming

[5] https://www.geeksforgeeks.org/what-is-lexical-scope-and-lexical-closures-in-dart/

[6] https://stackoverflow.com/questions/58966090/is-dart-pass-by-reference

[7] https://groups.google.com/a/dartlang.org/g/misc/c/jyn35RSuT9Q

[8] https://stackoverflow.com/questions/58568295/what-are-the-type-primitives-in-dart-flutter

[9] https://sarunw.com/posts/dart-parameters/

[10] https://dart-tutorial.com/functions/function-parameter-in-dart/

[11] https://stackoverflow.com/questions/71048808/what-is-closure-in-functions

[12] https://o7planning.org/14061/dart-closures