CS 315

Programming Languages

Section 3

Homework 3

Ömer Oktay Gültekin

21901413

2022-2023 / Fall Semester

SECTION 1: Code Explanations

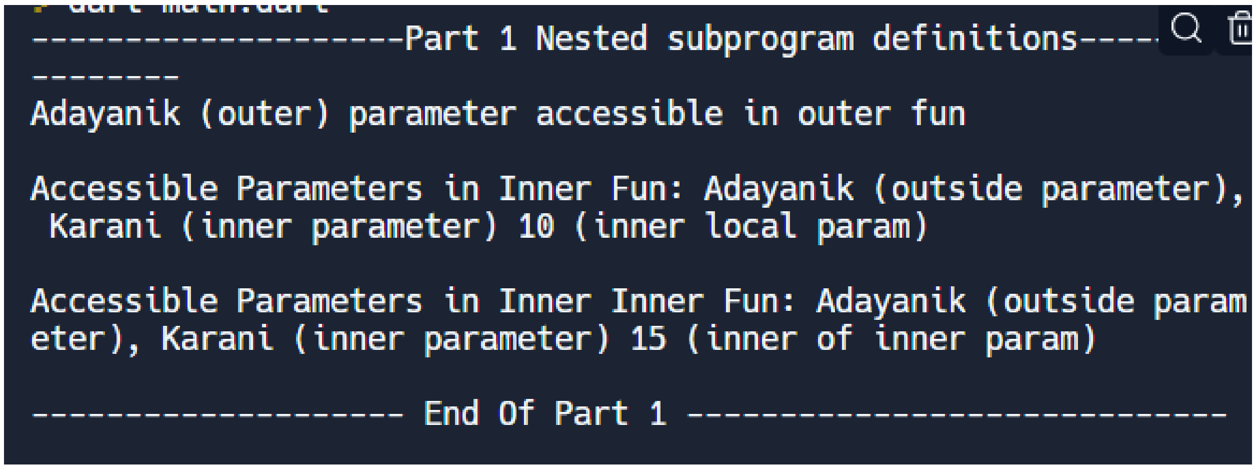
Most of the explanations is given in the comments, additional explanations are written in the report when necessary.

Part 1:

Code:

```
void main() {
  // --------------------Part 1 Nested subprogram definitions---------
--------
  print(
      "--------------------Part 1 Nested subprogram definitions-------
----------");
  void outerFun(String param) {
    print(param + " (outer) parameter accessible in outer fun");
    print("");

    // Subprogram 1
    void innerFun(String y) {
      int x = 10;
      print(
          "Accessible Parameters in Inner Fun: $param (outside
parameter), $y (inner parameter) $x (inner local param)");
      print("");

      // Subprogram 2
      void innerInnerFun(var z) {
        print(
            "Accessible Parameters in Inner Inner Fun: $param (outside
parameter), $y (inner parameter) $z (inner of inner param)");
        print("");
      }

      innerInnerFun(15);
    }

    // Not Accessible Here
    //innerInnerFun(10);
    innerFun("Karani");
  }
  // Not Accessible Here
  //innerInnerFun(10);
  //innerFun("Karani");

  outerFun("Adayanik");
  print("-------------------- End Of Part 1 -------------------------
---");
  print("");
  // ------------------------- End Of Part 1 -------------------------
-------------
```

Output:



```
---------------------Part 1 Nested subprogram definitions-----
--------
Adayanik (outer) parameter accessible in outer fun

Accessible Parameters in Inner Fun: Adayanik (outside parameter),
 Karani (inner parameter) 10 (inner local param)

Accessible Parameters in Inner Inner Fun: Adayanik (outside param
eter), Karani (inner parameter) 15 (inner of inner param)

--------------------- End Of Part 1 -----------------------------
```

Subprogram Definition in Dart is allowed with the condition that the subprograms are only accessible in the outer function’s scope (i.e not accessible in main). For subprograms static scoping is used and subprograms can access outer functions variables, but the reverse is not true.

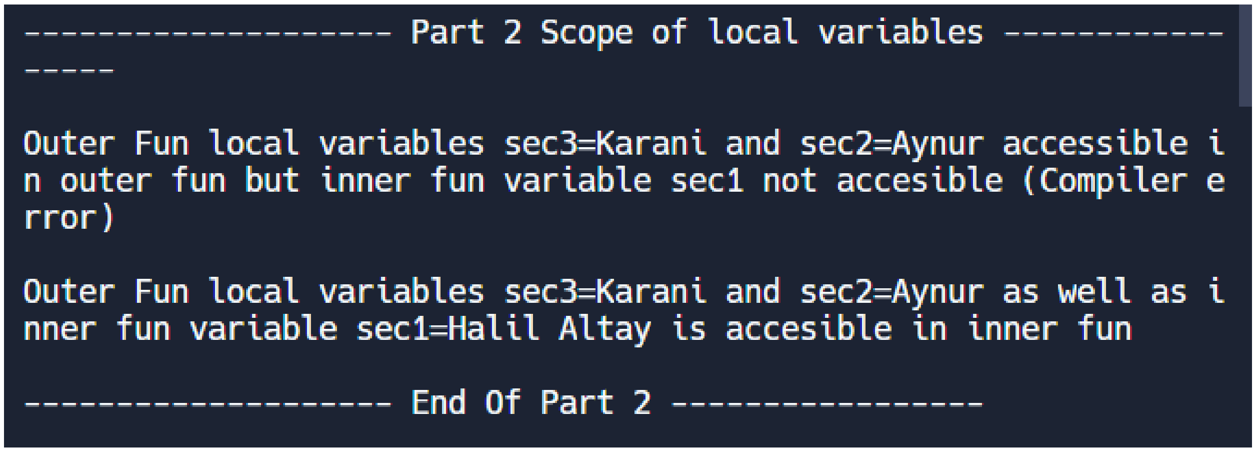
Part 2:

Code:

```
// -------------------- Part 2 Scope of local variables --------------
---
  print(
      "-------------------- Part 2 Scope of local variables ----------
-------");
```

```
  print("");

  void outerFun2(String param1, String param2) {
    String sec3 = param1;
    String sec2 = param2;

    print(
        "Outer Fun local variables sec3=$sec3 and sec2=$sec2
accessible in outer fun but inner fun variable sec1 not accesible
(Compiler error)");
    print("");

    // Since above inner fun is not in here, declaration with the same
name is allowed
    void innerFun(String param3) {
      String sec1 = param3;

      print(
          "Outer Fun local variables sec3=$sec3 and sec2=$sec2 as well
as inner fun variable sec1=$sec1 is accesible in inner fun");
      print("");
    }

    innerFun("Halil Altay");

    //print("Sec1 lecturer=$sec1") -- compile error because of illegal
statement since sec1 is not in this referencing environment
  }

  // Testing outer fun
  outerFun2("Karani", "Aynur");

  print("-------------------- End Of Part 2 -----------------");
  print("");

  // ---------------------------- End Of Part 2 ----------------------
----------------
```

Output:

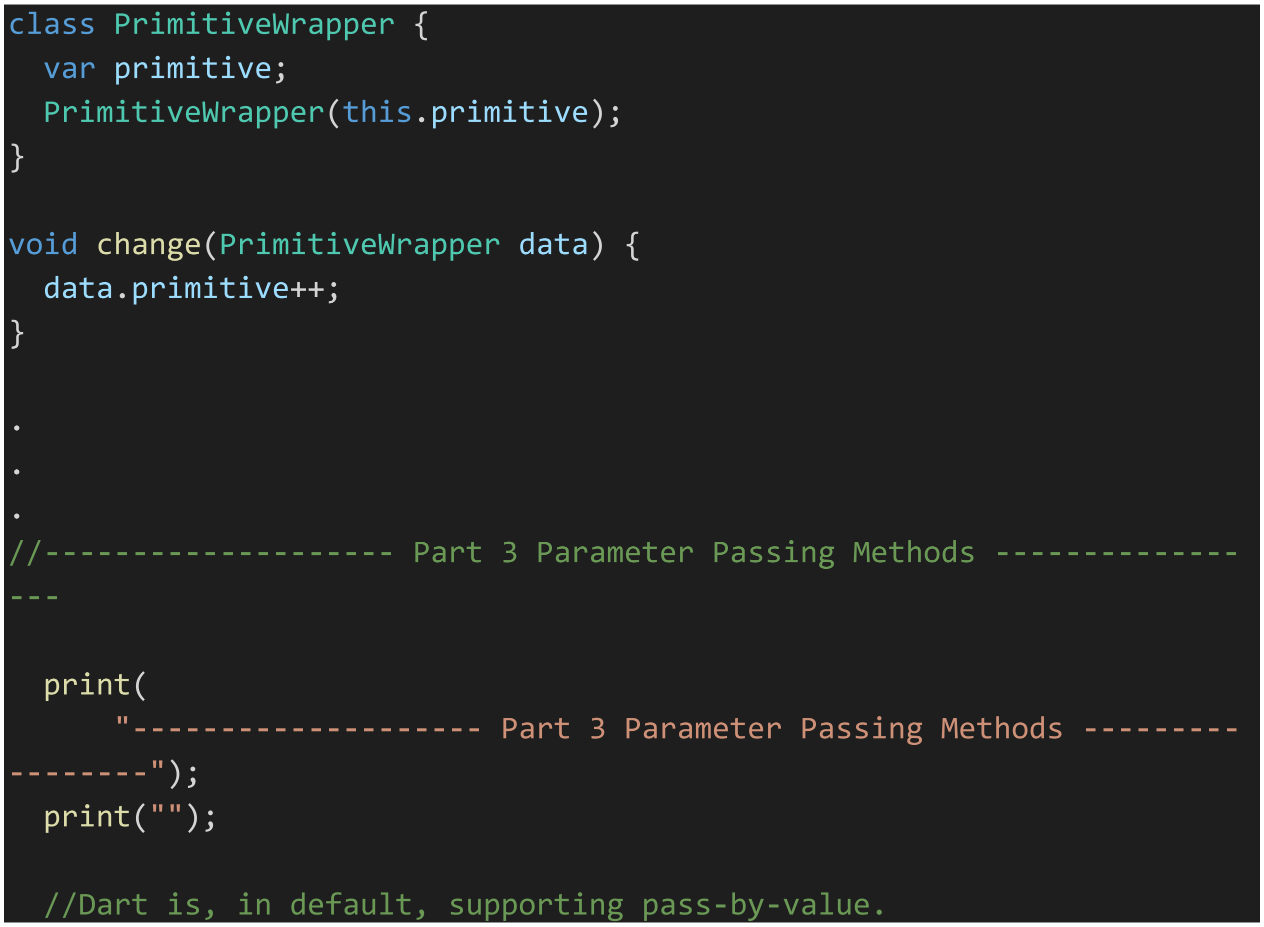


```
-------------------- Part 2 Scope of local variables ------------
-----

Outer Fun local variables sec3=Karani and sec2=Aynur accessible i
n outer fun but inner fun variable sec1 not accesible (Compiler e
rror)

Outer Fun local variables sec3=Karani and sec2=Aynur as well as i
nner fun variable sec1=Halil Altay is accesible in inner fun

-------------------- End Of Part 2 -----------------
```

Static scoping is used in Dart. (See part1 explanation for more info.)

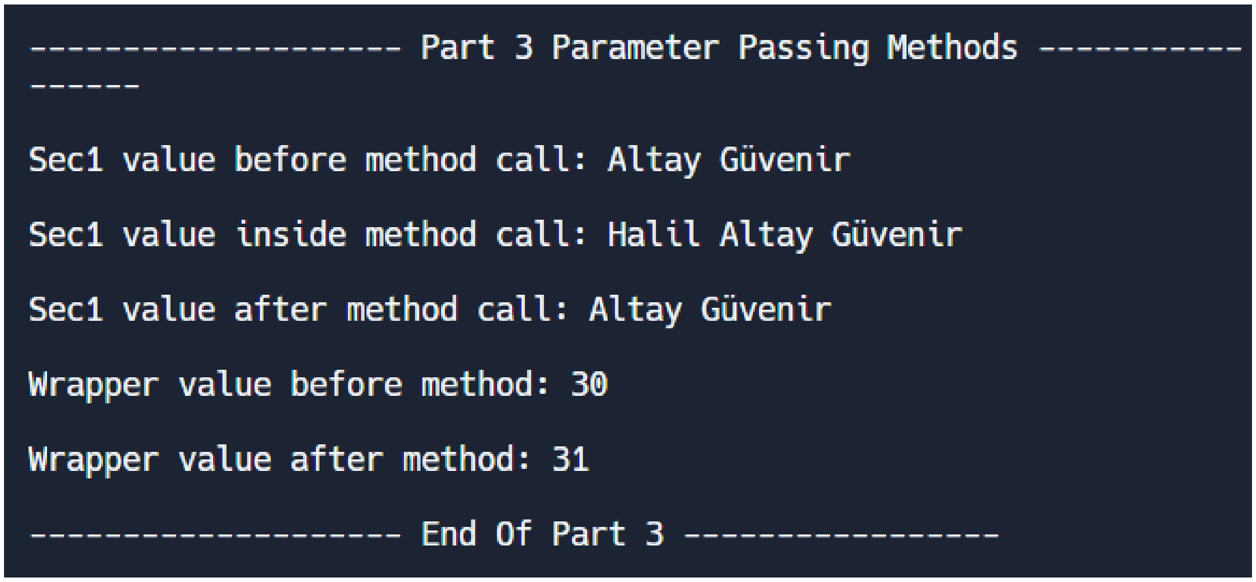
Part 3:

Code:



```
class PrimitiveWrapper {
  var primitive;
  PrimitiveWrapper(this.primitive);
}

void change(PrimitiveWrapper data) {
  data.primitive++;
}

.
.
.
//-------------------- Part 3 Parameter Passing Methods -------------
---


  print(
      "-------------------- Part 3 Parameter Passing Methods ---------
--------");
  print("");

  //Dart is, in default, supporting pass-by-value.
  String sec1 = "Altay Güvenir";

  void part3Fun(String param) {
    param = "Halil Altay Güvenir";
    print("Sec1 value inside method call: " + param);
    print("");
  }

  // Primitive types are passed by pass-by-value
  print("Sec1 value before method call: " + sec1);
  print("");

  part3Fun(sec1);
  print("Sec1 value after method call: " + sec1);
  print("");

  // Primitive types can be passed into wrappers to simulate pass-by-
reference
  var sec1Attendance = new PrimitiveWrapper(30);
  print("Wrapper value before method: ${sec1Attendance.primitive}");
// 30
  print("");

  change(sec1Attendance);
  print("Wrapper value after method: ${sec1Attendance.primitive}"); //
31
  print("");

  print("-------------------- End Of Part 3 -----------------");
  print("");

  // ---------------------------- End Of Part 3 ----------------------
----------------
```

Output:



```
-------------------- Part 3 Parameter Passing Methods -----------
------

Sec1 value before method call: Altay Güvenir

Sec1 value inside method call: Halil Altay Güvenir

Sec1 value after method call: Altay Güvenir

Wrapper value before method: 30

Wrapper value after method: 31

-------------------- End Of Part 3 -----------------
```

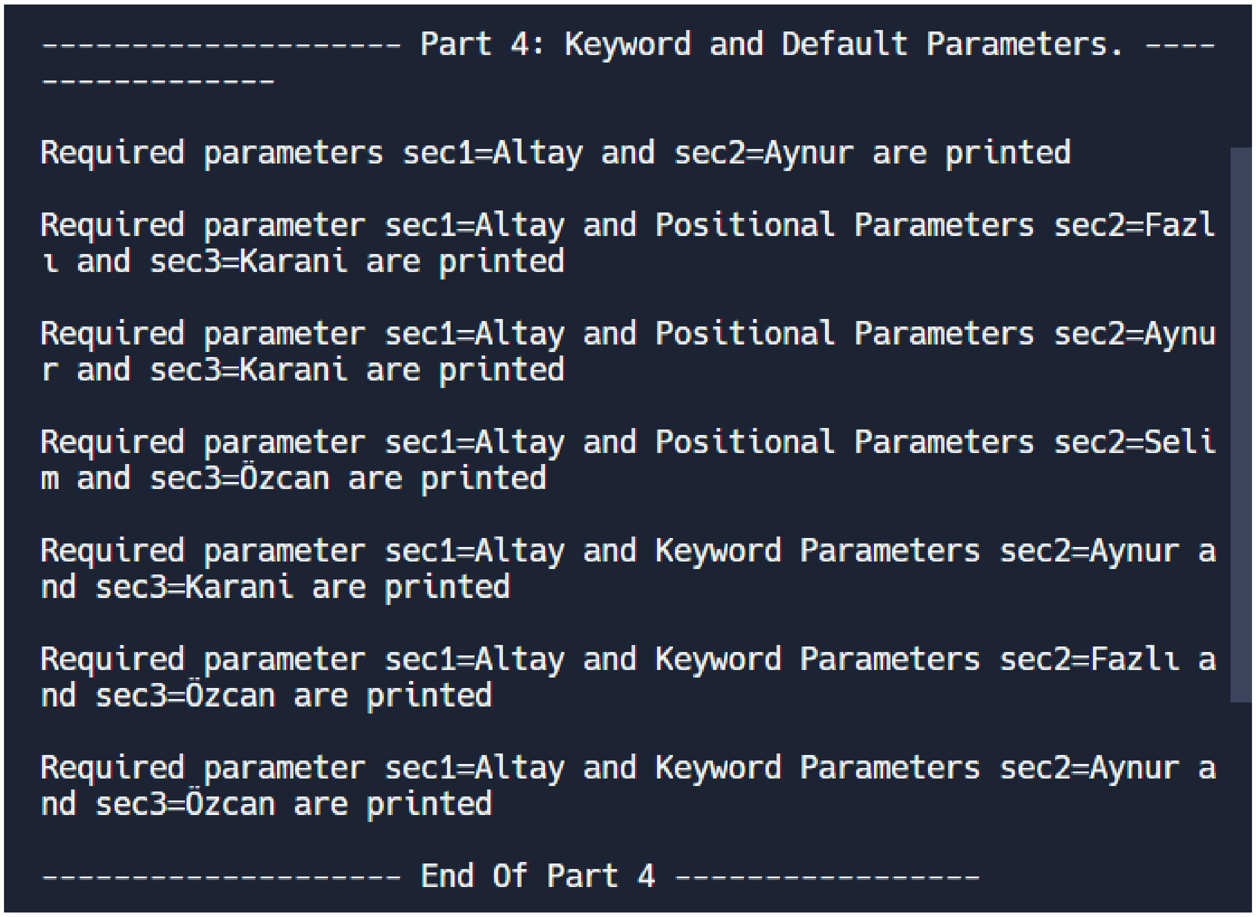
All parameters are passed by pass-by-value in the Dart (all variable types are actually Object in the Dart). A special method includes creating Wrapper for types are used to simulate pass-by-reference in the Dart.

Part 4:

Code:

```
//-------------------- Part 4: Keyword and Default Parameters. -------
----------

  print(
      "-------------------- Part 4: Keyword and Default Parameters. --
---------------");
  print("");

  //Optional parameters can be named (keyword) and positional
parameters in dart

  void requiredParamMethod(String sec1, String sec2) {
    // sec1 and sec2 are required parameters and can not have default
values, caller should always give values in method call, otherwise
there will be compilation error.
```

```
    print("Required parameters sec1=$sec1 and sec2=$sec2 are
printed");
    print("");
  }

  requiredParamMethod("Altay", "Aynur");
  // requiredParamMethod("Aynur"); error

  // Optional Positional Parameters Syntax
  // Here sec2 has a default value which can

  void requiredAndPositionalParamMethod(String sec1,
      [String sec2 = "Aynur", String sec3 = "Karani"]) {
    print(
        "Required parameter sec1=$sec1 and Positional Parameters
sec2=$sec2 and sec3=$sec3 are printed");
    print("");
  }

  requiredAndPositionalParamMethod(
      "Altay", "Fazlı"); // Fazlı hoca is sec2 lecturer now

  requiredAndPositionalParamMethod(
      "Altay"); // Sec2 and Sec3 uses their default values

  requiredAndPositionalParamMethod("Altay", "Selim",
      "Özcan"); // Sec2 and Sec3 lecturers are changed (One cannot
change sec3 lecturer without changing sec2 lecturer in positional
parameters)

  // Optional Named (Keyword) Parameters Syntax

  void requiredAndKeywordParamMethod(String sec1,
      {String sec2 = "Aynur", String sec3 = "Karani"}) {
    print(
        "Required parameter sec1=$sec1 and Keyword Parameters
sec2=$sec2 and sec3=$sec3 are printed");
    print("");
  }
  requiredAndKeywordParamMethod(
      "Altay"); // Sec2 and Sec3 uses default lecturers

  requiredAndKeywordParamMethod("Altay",
      sec3: "Özcan",
      sec2:
          "Fazlı"); // Order of the parameters is not important for
keyword parameters

  requiredAndKeywordParamMethod("Altay",
      sec3:
          "Özcan"); // Caller does not have to pass all keyword
parameters before sec3 as opposed to positional parameters (i.e
parameter passing is not restricted to positions )

  // Not allowed (required params must be before keyword params)
  // void requiredAndKeywordParam2Method({String sec2 = "Aynur",
String sec3 = "Karani"}, String sec1) {}

  // Not allowed (required params must be before positional params)
  // void requiredAndKeywordParam2Method([String sec2 = "Aynur",
String sec3 = "Karani"], String sec1) {}

  print("-------------------- End Of Part 4 -----------------");
  print("");
```
Output:

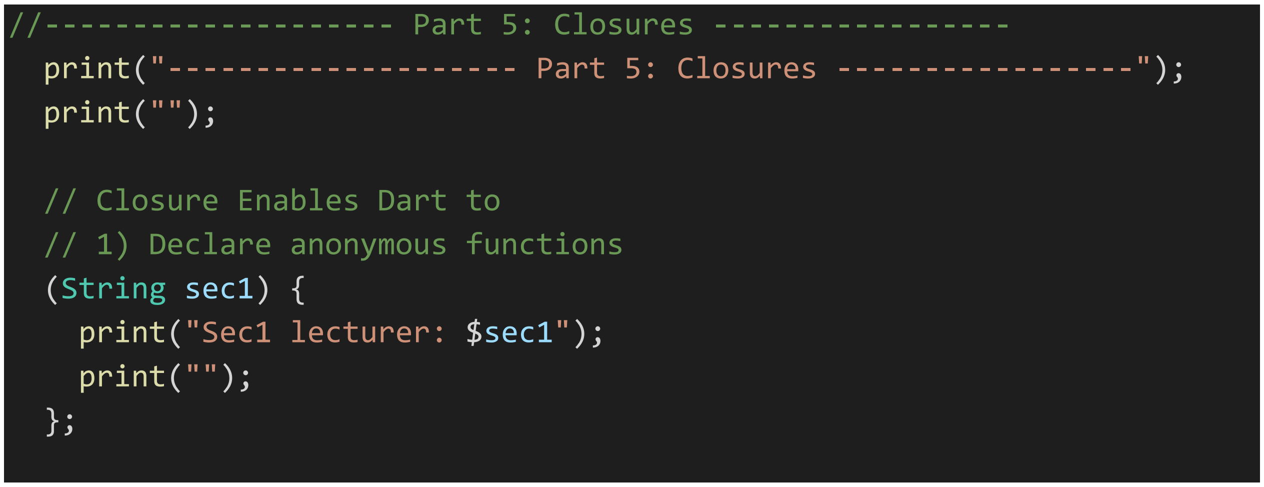


```
-------------------- Part 4: Keyword and Default Parameters. ----
-------------

Required parameters sec1=Altay and sec2=Aynur are printed

Required parameter sec1=Altay and Positional Parameters sec2=Fazl
ı and sec3=Karani are printed

Required parameter sec1=Altay and Positional Parameters sec2=Aynu
r and sec3=Karani are printed

Required parameter sec1=Altay and Positional Parameters sec2=Seli
m and sec3=Özcan are printed

Required parameter sec1=Altay and Keyword Parameters sec2=Aynur a
nd sec3=Karani are printed

Required parameter sec1=Altay and Keyword Parameters sec2=Fazlı a
nd sec3=Özcan are printed

Required parameter sec1=Altay and Keyword Parameters sec2=Aynur a
nd sec3=Özcan are printed

-------------------- End Of Part 4 -----------------
```

Optional keyword and positional arguments are declared with special notation in Dart.

Part 5:

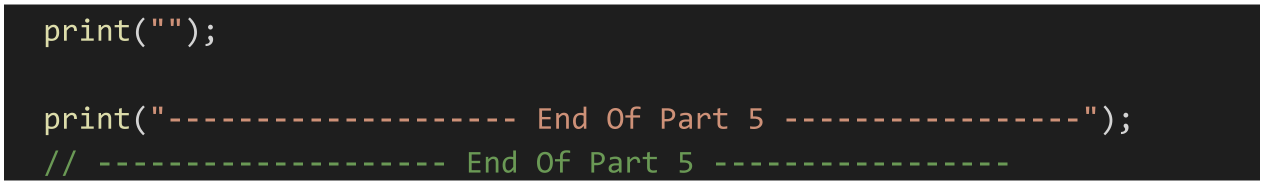
Code:



```
//-------------------- Part 5: Closures -----------------
  print("-------------------- Part 5: Closures -----------------");
  print("");

  // Closure Enables Dart to
  // 1) Declare anonymous functions
  (String sec1) {
    print("Sec1 lecturer: $sec1");
    print("");
  };
```

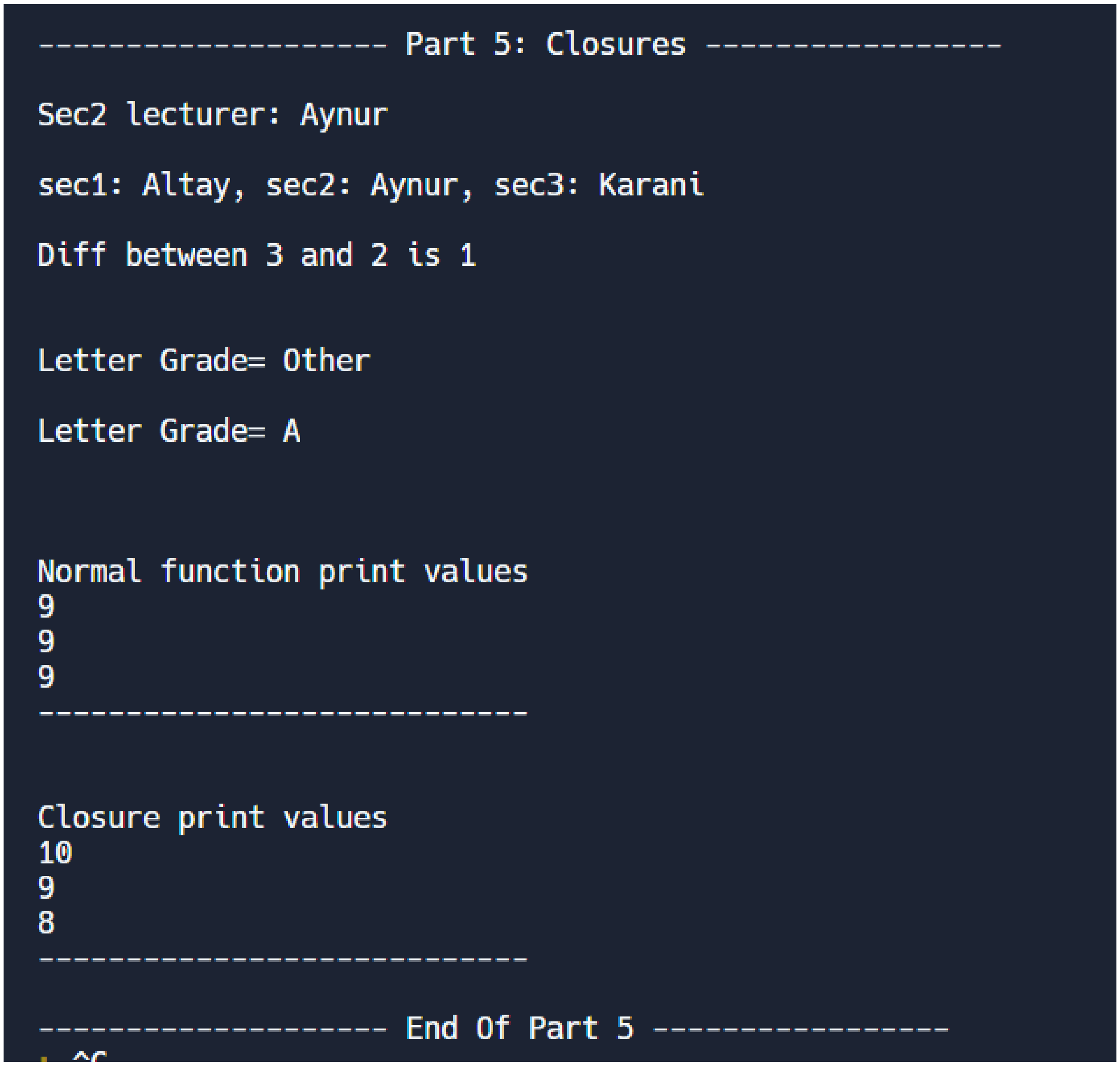
```dart
// 2) Use Anonymous functions directly
(String sec2) {
  print("Sec2 lecturer: $sec2");
  print("");
}("Aynur");

// 3) Assign alias variable to function
var printLecturers = (var sec1, var sec2, var sec3) {
  print("sec1: $sec1, sec2: $sec2, sec3: $sec3");
  print("");
};

printLecturers("Altay", "Aynur", "Karani");

// 4) Arrow Syntax Function Definitions
var difference = (var bigNum, var smallNum) => bigNum - smallNum;

print("Diff between 3 and 2 is ${difference(3, 2)}");
print("");

// 5) Use Functions in Parameter
void calculateGPA(int grade, String getLetterGrade(int grade)) {
  print("Letter Grade= ${getLetterGrade(grade)}");
  print("");
}

var getGermanLetterGrade = (int grade) {
  return grade > 95 ? "A" : "Other";
};

var getMathLetterGrade = (int grade) {
  return grade > 85 ? "A" : "Other";
};

print("");
calculateGPA(90, getGermanLetterGrade);
calculateGPA(90, getMathLetterGrade);
print("");
```

```
  // 6) Using outer function local variables even after they should be
not accessible anymore

  // Normal subprogram definitions (number are not accesible in main
scope (only the state of the number is accessible))
  int subtract() {
    var number = 10;
    void minus() {
      number--;
    }

    minus();
    return number;
  }

  print("");
  print("Normal function print values");
  print(subtract());
  print(subtract());
  print(subtract());
  print("----------------------------");
  print("");

  // Closure subprogram definition (key point is returning the
function)

  Function() subtractClosure() {
    var number = 10;
    return () => number--;
  }

  var subtractor = subtractClosure();

  print("");
  print("Closure print values");
  print(subtractor());
  print(subtractor());
  print(subtractor());
  print("----------------------------");
```



```
  print("");

  print("-------------------- End Of Part 5 -----------------");
  // -------------------- End Of Part 5 -----------------
```

Output:



```
-------------------- Part 5: Closures -----------------

Sec2 lecturer: Aynur

sec1: Altay, sec2: Aynur, sec3: Karani

Diff between 3 and 2 is 1


Letter Grade= Other

Letter Grade= A



Normal function print values
9
9
9
----------------------------


Closure print values
10
9
8
----------------------------

-------------------- End Of Part 5 -----------------
```

In Dart, Closures allows to define anonymous functions and using them as if they are special type, Function. The key advantage of the closures is to use the local variables of the functions that normally destroyed when the function ends via returning the function in the function definition. The difference between the closure and the normal subprogram definitions is, in closures, the return value of the function is the subfunction whereas in normal functions, the subfunction is located in the body of the outer function that is not accessible outside of the function. Closure is the implementation of the callbacks which exists in some other programming languages like JS.

SECTION 2: Evaluation

The subprogram definition of the Dart is highly writable and readable since it uses static scope that is used in most programming languages. Also, not accessing the subprograms outside the outer function increase readability since users don't have to trace the usage of the subprograms outside the function.

The scope of the local variables are highly readable since the scope is following the program flow and reader can directly understand the referencing environment local variable use. It is highly writable because of the same logic.

Passing all parameters by value is highly readable and writable since the writer should not memorize each specific case for different types of variables and reader can always assume that given parameter to the function will never modified in the code unless wrappers are used which is easily understood when used in the code.

Writing of the keywords and default parameters is less writable than other languages since the writer should know special syntax ([] used for positional parameters and {} used for keyword parameters). Also, they are less readable than other languages since reader should look at how each parameter is defined separately but for example in Python, any parameter can be positional, keyword or have default value which is highly readable.

The closures enhances writability of the language since they allows writer to use anonymous functions, arrow functions, to define variables with function type and callback functionality. Also, Closure syntax, I think, is readable since it resembles the other languages implementation of the callbacks.

SECTION 3: LEARNING STRATEGY

Since I was writing mobile apps with flutter, which uses Dart as a programming language, I know most of the information that can be found in the report. When necessary I use the information found on the internet which is mostly online tutorials and general tour of Dart which is on the Dart official website. Other than these, I didn't use any additional source which means I didn't have any personal contact. Online links I used in this homework is as follows:

https://stackoverflow.com/questions/18258267/is-there-a-way-to-pass-a-primitive-parameter-by-reference-in-dart

https://social.msdn.microsoft.com/Forums/en-US/28235a32-7c9b-4d22-b95b-b680533ed39c/nested-function-and-closure?forum=asphtmlcssjavascript

https://o7planning.org/14061/dart-closures

https://medium.com/@MrArc/dart-variables-7dbcc943448d

https://medium.com/vijay-r/parameter-passing-types-dart-16dcc5cad9a6

https://github.com/dart-lang/language/issues/1911

https://dart.dev/guides/language/language-tour

https://dart.dev/search?cx=011220921317074318178%3A_yy-tmb5t_i&ie=UTF-8&hl=en&q=local+variable

https://www.educative.io/answers/how-to-create-a-nested-function-in-dart

Online Compiler I have use (I made workspaces public so you can see the code via link):

I have used a replit code environment to test my codes.
https://replit.com/@MrKty/gultekinomeroktayhw3dart#main.dart