CS315

HW3

Subprograms in Dart

Elifsena ÖZ

22002245

Section 1

Contents

1) Nested Subprogram Definitions

Below code defines two subprograms. Subprogram nested_outer() has nested_inner() inside it. When the last line of nested_outer() is executed, nested_inner() is called. Here we can see that dart allows definition and execution of nested subprograms [1].

Code:

```
// Nested Subprogram Definitions
  void nested_outer() {
    void nested_inner() {
      print("Function nested_inner() nested inside nested_outer()");
    }
    print("Function nested_outer()");
    nested_inner();
  }

  print("~~ Nested Subprogram Definitions ~~");
  nested_outer();
```

Output:

```
~~ Nested Subprogram Definitions ~~
Function nested_outer()
Function nested_inner() nested inside nested_outer()
```

2) Scope of Local Variables

The code segment below contains two nested functions named scope_outer() and scope_inner(). Each function as well as the main() function has a variable defined in it. For the main() x is defined and initialized as 3, for scope_outer() y is defined and initialized as 4, for scope_inner() z is defined and initialized as 5. First the value of x is printed before all operations. Then, the value of x and y is printed in scope_outer() before scope_inner(). All values are printed in scope_inner() followed by the values of x and y in scope_outer(). Lastly x will be printed after scope outer. Both scope_inner() and scope_outer() modify the local variable of their parents with addition operations. This printing sequence will clearly show how the variables have been modified after each function.

With the output of this code, we can see that the local variables of parent functions can be reached by child functions. They can also be modified.

Code:

```
// Scope of Local Variables
  print("\n~~ Scope of Local Variables ~~");

  int x = 3;

  void scope_outer() {
```

```
  int y = 4;

  void scope_inner() {
    int z = 5;
    print("In scope_inner x is $x, y is $y, z is $z");

    // Scope inner adds z to x and y values
    x += z;
    y += z;
  }

  print("Before scope_inner() in scope_outer x is $x, y is $y");
  scope_inner();
  print("After scope_inner() in scope_outer x is $x, y is $y");

  // Scope outer adds y to x value
  x += y;
}

print("Before scope_outer x is $x");
scope_outer();
print("After scope_outer x is $x");
```

Output:

```
~~ Scope of Local Variables ~~
Before scope_outer x is 3
Before scope_inner() in scope_outer x is 3, y is 4
In scope_inner x is 3, y is 4, z is 5
After scope_inner() in scope_outer x is 8, y is 9
After scope_outer x is 17
```

3) Parameter Passing Methods

Below code passes i as a parameter to param_pass() method. This method adds 4 to the parameter and returns the parameter. By the output of the following code segment we can see that the value of I is passes to the function but cannot be modified outside the function. Operations can be done on the value. However, they will only affect the result that the function returns (in this case returned_param variable) [2].

```
// Parameter Passing Methods
print("\n~~ Parameter Passing Methods ~~");

int i = 1;

int param_pass(int x) {
  // param_pass() adds 4 to the parameter
  x += 4;
```

```
    print("In param_pass() parameter is $x");
    return x;
  }

  print("Before param_pass() parameter is $i");
  int returned_param = param_pass(i);
  print("param_pass() returns $returned_param");
  print("After param_pass() parameter is $i");
```

Output:

```
~~ Parameter Passing Methods ~~
Before param_pass() parameter is 1
In param_pass() parameter is 5
param_pass() returns 5
After param_pass() parameter is 1
```

4) Keywords and Default Parameters

Below code shows three functions that all take two parameters: foo(), boo(), and moo(). In foo() both parameters are required, and the function will not execute if the parameters are not given in calling. In boo() first parameter is not required and will default to null whereas the second parameter is also not required and will default to the value given in function declaration (in this case “default” string). In moo() first parameter is not required and has default, second parameter is explicitly declared as required. In following lines you can see the different callings on functions and the output. In the output words written in ALL CAPS are given PARAMETERS and words written in all lowercase are default values (ex. null and default) [3].

Code:

```
// Keywords and Default Parameters
  print("\n~~ Keywords and Default Parameters ~~");
  print("!Caution! \nWords written in ALL CAPS are given PARAMETERS and words
written in all lowercase are default values.\n");

  // both parameters are required
  void foo(String isRequired1, String isRequired2) {
    print("In foo first parameter is $isRequired1, second parameter is
$isRequired2");
  }

  // first parameter is not required, second is not required and has default
  void boo({String? notRequired, String hasDefault = "default"}) {
    print("In boo first parameter is $notRequired, second parameter is
$hasDefault");
  }
```

```
  // first parameter is not required and has default, second parameter is
explicitly required
  void moo({String hasDefault = "default", required String isRequired}) {
    print("In moo first parameter is $hasDefault, second parameter is
$isRequired");
  }

  foo("REQUIRED", "REQUIRED");

  boo();
  boo(notRequired:"NOT_REQUIRED", hasDefault:"NOT_REQUIRED");
  boo(notRequired:"NOT_REQUIRED");

  moo(isRequired:"REQUIRED");
```

Output:

```
~~ Keywords and Default Parameters ~~
!Caution!
Words written in ALL CAPS are given PARAMETERS and words written in all
lowercase are default values.

In foo first parameter is REQUIRED, second parameter is REQUIRED
In boo first parameter is null, second parameter is default
In boo first parameter is NOT_REQUIRED, second parameter is
NOT_REQUIRED
In boo first parameter is NOT_REQUIRED, second parameter is default
In moo first parameter is default, second parameter is REQUIRED
```

5) Closures

In below code 3 closures have been declared. First closure is not assigned to a variable and directly called after definition. Second closure is assigned to a variable and called using the variable. This closure is assigned to a variable and uses the “=>” for simplicity and code writability [4].

Code:

```
  // Closures
  print("\n~~ Closures ~~");

  // closure directly called
  (String hw) {
    print("Done with $hw!");
  }("hw3");

  // closure as variable
  var closure = (String task) {
    print("No! You still have to do the $task!");
  };
```

```
  closure("report");

  // closure with arrow
  var closure2 = (String mood) => print("I feel $mood.");
  closure2("sad");
```

Output:

```
~~ Closures ~~
Done with hw3!
No! You still have to do the report!
I feel sad.
```

6) Readability of Dart Syntax

Nested subprograms are easy to detect and understand in dart. Scope of local variables are not confusing as the variables can be read from the parent scope. Parameters are passed with pass by value which decreases the readability of the language as people may expect that the operations on the parameter are permanent. Giving default parameters, using “?” to make parameter non required and explicitly using “required” keywords make it really easy for the reader to understand what the function expects as parameters. The default closure statements are a little confusing and hard to understand. However, the usage of “=>” clears up the statements and makes it concise. It can only be used for single line closures. Therefore, in my opinion the closures are readable as long as they are single line.

7) Writability of Dart Syntax

Nested subprograms and scope of local variables were very straightforward in Dart. I did not have any problems with the design of the language in these topics. Parameter passing is not easily done as the parameters are passed by value and not reference, which decreases the writability of the language. For keywords and default parameters as the keywords and symbols are very commonly used therefore, would increase the writability of the language. Closures are a little confusing in terms of writability as well as readability. The placement of semicolons and parentheses are important and require a lot of attention except for the single line closures which can be used with “=>” operator. I believe that they generally reduce the writability of the language.

8) Learning Strategies

As it was with any other homework for this class, this homework required a lot of research and a long process of trial and error. In this process I got a lot of help from the documentation of the language as well as some popular programming websites such as GeeksforGeeks and StackOverflow. I also used Dartpad as an online compiler [5]. This homework was a process in programming that I always came across: learning by yourself.

From my experience until now programming is not a profession that can be learned fully in school. It takes a lot of determination and personal motivation to learn a new programming language.

9) References

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

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

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

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

[5] https://dartpad.dev/