

## Part 1) Design Issues:

### Nested Subprogram Definitions:

In Dart, nesting functions as subprograms is possible, but it is impossible to do the same for procedures.

Below, a simple example is given. When the function foo1 is called, it will call a nested function called foo2. Due to the scope of nested subprograms in Dart, foo2 can see the local variable of foo1. Hence, it will print whatever value is stored in i.

```
print("Example1:");
void foo1() {
    int i = 1;
    void foo2() {
        print(i);
    }
    foo2();
}
foo1();
```

*Figure 1: Nesting Functions Example 1*

Here is a more complex algorithm for understanding what nested functions do and what they don't by default. First, a variable j is declared, without any initial value, so it is null by default. Then, foo3 function is called. Since pass-by-value is used, it is not expected for the outer variable j to be anything but null after calling foo3. When foo3 is called, the value j in the function is null. After the first print statement, the nested function sub1 is given. After that when the second print statement is called, the value of j is still null, as you need to call the nested function to execute it. Nested function sub1 gives 999 value to j and returns it, hence the third print function in the foo3 body prints 999. The one later print statement also prints 999, as the value of j changed when sub1 is called. Then, when the print statement coming after the execution of foo3 is executed, null is printed, as pass-by-value is used.

```
print("Example2:");
var j;
void foo3(var j) {
    print("foo3 initial j:" + j.toString());
    sub1 () {
        j = 999;
        return j;
    }
    print("foo3 before sub1 j:" + j.toString());
    print("sub1() j:" + sub1().toString());
    print("foo3 after sub1 j:" + j.toString());
}
foo3(j);          //calling foo3
print("after foo3 j:" + j.toString());
```

*Figure 2: Nesting Functions Example 2*

### **Scope of Local Variables:**

The scope of a local variable is the block it is defined. This means that when the block terminates, the local variable is destroyed. And this means that in an outer block, it is not possible to reach the local variable of an inner block.

```
print("Example1:");
int i = 1;
{
    int i = 2;
    print("middle i:" + i.toString());
    {
        int i = 3;
        print("inner i:" + i.toString());
    }
    print("middle i:" + i.toString());
}
print("outer i:" + i.toString());
print("\n");
```

*Figure 3: Scope of Local Variables Example 1*

However, if no local variable with the same name is declared, it is possible to reach an outer block's local variable.

```
print("Example2:");
i = 1; // i is already declared in the previous example, hence, I reassigned a value
{
    int j = 2;
    {
        int k = 3;
        print("inner i:" + i.toString());
        print("inner j:" + j.toString());
        print("inner k:" + k.toString());
    }
    print("middle i:" + i.toString());
    print("middle j:" + j.toString());
}
print("outer i:" + i.toString());
```

*Figure 4: Scope of Local Variables Example 2*

### **Parameter Passing Methods(Positional parameter passing excluded):**

It is possible to use pass-by-value in Dart. Primitives are pass-by-value in Dart. Below, an example of it is given. The variable's value will not change after the function is called as its value is passed.

```
print("Pass by value:");
void increment(int val) {
    val = val + 1;
}
int val = 4;
print("val before: " + val.toString());
increment(val);
print("val after: " + val.toString());
```

*Figure 5: Parameter Passing Methods, Pass by Value*

Pass-by-reference can't be used for primitive data types in Dart, however, encapsulating primitive data types with objects and using pass-by-reference is possible, as Dart is pass-by-reference for objects. Look to the beginning of the file to understand the object. Its initial value is 0. Since objects are pass-by-reference, the incrementObject function can increase its value, not its copy value[1].

```
print("\nPass by reference:");
void incrementObject(Integer integer) {
    integer.i++;
}
Integer someInt = new Integer();
print("object value before incrementing: " + someInt.i.toString());
incrementObject(someInt);
print("object value after incrementing: " + someInt.i.toString());
```

*Figure 6: Parameter Passing Methods, Pass by Reference*

It should be noted that I sort of bent the meaning of pass-by-reference and pass-by-value here[2]. Dart is, in reality, pass-by-value both for primitive data types and objects. However, when an object is passed as a parameter to a function, what really happens there is that its reference is passed as a value to the function. With this logic, many Object-Oriented languages are actually fully pass-by-value. With this logic, Dart is also always pass-by-value[2].

## **Keyword and Default Parameters:**

There are many ways to pass parameters to a function in Dart programming language. They will be covered here. Understanding positional parameters are necessary for understanding keyword and default parameters.

If positional parameters are used, then the result of the function is going to change when we put actual parameters in a different order. In the example below, we see that the result of a simple calculation changes when the order of actual parameters changes[3].

```
print("Example: Substraction");
int subtract(int a, int b) {
    return a - b;
}
int a = 10;
int b = 5;
print("10 - 5 = " + subtract(a,b).toString());
print("5 - 10 = " + subtract(b,a).toString());
```

*Figure 7: Keyword and Default Parameters, Example Substraction*

It is possible to give default values to OPTIONAL positional parameters in Dart. This means, if a variable is non-optional, it is not possible to give it an initial value. When the mult1 function is called the first time, no actual parameter is given to b. Hence, b takes value 100. This doesn't happen in the second call[3].

```
print("\nExample: mult1");
int mult1(int a, [int b=100]) {
    return a * b;
}
print("b takes initial value: " + mult1(a).toString());
print("b takes passed value: " + mult1(a,2).toString());
```

*Figure 8: Keyword and Default Parameters, Example Multiplication 1*

One thing worthy of noting about optional positional parameters is parameters on the left side have precedence over the right ones. What I mean is this: If we have two consecutive optional positional parameters, and we pass an actual parameter, Dart should determine which optional positional parameter will take the passed value and which one will take its initial value. The left one takes the passed value, and the ones on the right take initial values. Here is an example of it given below[3].

```
print("\nExample: mult2");
int mult2([int a = 2, int b = 3]) {
    return a * b;
}
//reassigning for demonstration purposes.
a = 10;
b = 5;
print("both take initial values: " + mult2().toString());           //both take initial values: prints 6.
print("b takes initial value: " + mult2(a).toString());             //a=10, b=3: prints 30
//print("Can't be done!: " + mult2(,a).toString());                 //invalid.
print("both take passed values: " + mult2(a,b).toString());         //a=10, b=5: prints 50
```

*Figure 9: Keyword and Default Parameters, Example Multiplication 2*

It is quite possible that a function takes many parameters. And if the list of this parameters get too long, it is likely that the order will be written wrong by the developer. This is an issue

both for readability and writability, I would say. Named parameters solve this issue: formal parameter names can be used to get actual parameters in any order. ? is used for null safety.

```
void subtract2({int? a, int? b}) {  
    print("$a - $b");  
}  
a = 999;  
b = 333;  
// Even if the order of passed parameters change, the output stays the same:999 - 333.  
subtract2(a: a, b: b);  
subtract2(b: b, a: a);  
subtract2(); //it is not mandatory to pass actual parameters here. for that, use "required"
```

Figure 10: Keyword and Default Parameters, Example for Named Parameters

It is possible to require to pass an actual parameter to a named parameter, using the "required" keyword.

```
void subtract3({required int a, int? b}) {  
    print("$a - $b");  
}  
a = 888;  
b = 222;  
//subtract3(); //wouldn't work.  
subtract3(a: a); //works.  
subtract3(a: a, b: b); //works.
```

Figure 11: Keyword and Default Parameters, Example for Keywords

### **Closures:**

Using closures, we can access to variables of a terminated scope. Here, the variable a is declared in scope of returnFunc, and since closure exists, the function that is stored as an object in someFunc variable can access variable in returnFunc even after it is terminated.

```
Function returnFunc = () {  
    int a = 0;  
    Function printSomething = () {  
        a++;  
        print(a);  
    };  
    return printSomething;  
};  
var someFunc = returnFunc();  
someFunc();  
someFunc();  
someFunc();
```

Figure 12: Closures

## Part 2) Readability and Writability of Subprogram Syntax

I didn't have much trouble both in terms of readability and writability in Dart. However, I think writing nested subprograms creates readability issues if they exist in a programming language, independent from the programming language. It can be hard to keep track of them if there are too many of them nested in each other.

## Part 3) Learning Strategy

I found searching about implementations of pass-by-value and pass-by-reference in Object Oriented programming languages helpful. I read a thread about Java in these issues, and it opened my mind about Dart[2].


I used the documentation of Dart to understand Named and Optional parameters better. Certain websites were useful for reaching knowledge fast. I also used some YouTube tutorials. All of these are given in the references.

I didn't use online compilers. Dart is installed on my Linux machine.

```
saturnine@pop-os:~/cs315/hw3$ dart --version
Dart SDK version: 2.18.4 (stable) (Tue Nov 1 15:15:07 2022 +0000) on "linux_x64"
```

*Figure 13: Dart on my Linux*

## References

1. <https://groups.google.com/a/dartlang.org/g/misc/c/jyn35RSuT9Q>
2. <https://stackoverflow.com/questions/40480/is-java-pass-by-reference-or-pass-by-value>
3. <https://dart.dev/guides/language/language-tour#parameters>
4. <https://dart-tutorial.com/functions/function-parameter-in-dart/>
5.  Dart Lexical Closures Tutorial (Functional Programming) #10.3