2019-2020 Fall

CS 315

Homework Assignment 1

Section: 3
Name: Munib Emre
Surname: Sevilgen
Student ID: 21602416

Dart

Example in Dart:

```
main (){
   var x = "x of main";
   var y = "y of main";

   foo() {
       // print("\nWhile entering foo: " + x + " " + y);
       // Error: Can't declare 'y' because it was already
       // used in this scope.
       print("\nWhile entering foo: " + x);
       x = "x of foo";
       var y = "var y of foo";
       print("In foo: " + x + ", " + y);

       // The order of the functions is changed because
       // otherwise there is an error:
       // Error: Method not found: 'foo3'.
       foo3() {
           print("\nWhile entering foo3: " + x + ", " + y);
           x = "x of foo3";
           y = "y of foo3";
           print("While exiting foo3: " + x + ", " + y);

       }

       // The order of the functions is changed because
       // otherwise there is an error:
       // Error: Method not found: 'foo2'.
       foo2() {
           print("\nWhile entering foo2: " + x + ", " + y);
           x = "x of foo2";
           y = "y of foo2";
           print("While exiting foo2: " + x + ", " + y);
           foo3();
       }

       foo1() {
           // print("\nWhile entering foo1: " + x + " " + y);
           // Error: Can't declare 'x' because it was already
           // used in this scope.
           print("\nWhile entering foo1: " + y);
           var x = "x of foo1";
           y = "y of foo1";
           print("In foo1: " + x + ", " + y);
```

```
            foo1_1() {
               print("\nWhile entering foo1_1: " + x + ", " + y);
               x = "x of foo1_1";
               y = "y of foo1_1";
               print("While exiting foo1_1: " + x + ", " + y);
            }

            foo1_1();
            print("After foo1_1 in foo1: " + x + ", " + y);

            foo2();
            print("While exiting foo1: " + x + ", " + y);
         }

         foo1();
         print("After foo1 in foo: " + x + ", " + y);
         foo2();
         print("While exiting foo: " + x + ", " + y);
      }

      foo();

      print("\nIn main: " + x + ", " + y);
}
```

The output of the example in Dart:

```
While entering foo: x of main
In foo: x of foo, var y of foo

While entering foo1: var y of foo
In foo1: x of foo1, y of foo1

While entering foo1_1: x of foo1, y of foo1
While exiting foo1_1: x of foo1_1, y of foo1_1
After foo1_1 in foo1: x of foo1_1, y of foo1_1

While entering foo2: x of foo, y of foo1_1
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting foo1: x of foo1_1, y of foo3
After foo1 in foo: x of foo3, y of foo3

While entering foo2: x of foo3, y of foo3
```

```
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting foo: x of foo3, y of foo3

In main: x of foo3, y of main
```

In Dart language, the variables declared in the outer scopes can be accessed and assigned to different values if there is not any new declaration with the same variable name in the local scope. The variables in the closer outer scope can be reached in the inner scopes. In the example, in foo function there is the new declaration of y but not x, then in the inner scopes and the current scope of foo, the x variable of main and the y variable of foo can be reached. The y of the main is hidden for the inner scopes. This can be also observable in the foo1 function. In foo1 function there is the new declaration of x but not y, then in the inner scopes and the current scope of foo, the y variable of foo and the x variable of foo1 can be reached. Also, foo1_1 can reach the x of the main and y of foo function. The variables in the outer scopes can be reached by not just one inner scope but all inner scopes.

Moreover, when the foo1 function calls the foo2 and then foo2 calls the foo3 function that is at the same scope with the foo1 that is the scope in foo, foo1, and foo2 functions change the x of the main and y of the foo function. Therefore there is the static scoping not dynamic scoping in the Dart language. These scoping rules make the Dart language more readable but it decreases the flexibility of the language.

JavaScript

Example in JavaScript:

```
<!DOCTYPE html>
<html lang="en">
<body>
<script>
x = "x of global";// Can't delete this line because Reference error in
foo
y = "y of global"

function foo() {
   console.log("\nWhile entering foo: " + x + ", " + y);
   x = "x of foo";
   var y = "var y of foo";
   console.log("In foo: " + x + ", " + y);

   function foo1() {
       console.log("\nWhile entering foo1: " + x + ", " + y);
       var x = "x of foo1";
       y = "y of foo1";
       console.log("In foo1: " + x + ", " + y);
```

```
        function foo1_1() {
            console.log("\nWhile entering foo1_1: " + x + ", " + y);
            x = "x of foo1_1";
            y = "y of foo1_1";
            console.log("While exiting foo1_1: " + x + ", " + y);
        }

        foo1_1();
        console.log("After foo1_1 in foo1: " + x + ", " + y);
        foo2();
        console.log("While exiting foo1: " + x + ", " + y);
    }

    function foo2() {
        console.log("\nWhile entering foo2: " + x + ", " + y);
        x = "x of foo2";
        y = "y of foo2";
        console.log("While exiting foo2: " + x + ", " + y);
        foo3();
    }

    function foo3() {
        console.log("\nWhile entering foo3: " + x + ", " + y);
        x = "x of foo3";
        y = "y of foo3";
        console.log("While exiting foo3: " + x + ", " + y);
    }

    foo1();
    console.log("After foo1 in foo: " + x + ", " + y);
    foo2();
    console.log("While exiting foo: " + x + ", " + y);
}

foo();

console.log("\nIn global: " + x + ", " + y);

</script>
</body>
</html>
```

The output of the example in JavaScript:

```
While entering foo: x of global, undefined
In foo: x of foo, var y of foo
```

```
While entering foo1: undefined, var y of foo
In foo1: x of foo1, y of foo1

While entering foo1_1: x of foo1, y of foo1
While exiting foo1_1: x of foo1_1, y of foo1_1
After foo1_1 in foo1: x of foo1_1, y of foo1_1

While entering foo2: x of foo, y of foo1_1
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting foo1: x of foo1_1, y of foo3
After foo1 in foo: x of foo3, y of foo3

While entering foo2: x of foo3, y of foo3
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting foo: x of foo3, y of foo3

In global: x of foo3, y of global
```

In JavaScript, by the ES6 and later versions of the Javascript that is used in the latest versions of the browsers, the variables should be declared with var keyword before they are used. If not, it gives a reference error. In the old versions, when the assignment is tried without the declaration with var before, it looks for the outer scopes. If there is not any declaration in the outer scopes, it automatically declares the variable at the global scope. Therefore, I need to declare the x and y in the global scope to use the x in the foo function.

The variables in the outer scopes can be reached and changed from the inner scopes in Javascript if there is not any new declaration with the same name in the local scope as same as the Dart. In the example, there is not any declaration of x in foo, then it can print the x in the global. However, y in the foo function is declared after printing “While entering foo: x of global, undefined:”. The y is printed as undefined because it is assigned later. The declaration of the variables is moved to the top of the scope by the JavaScript but not assigned to a value.

There is the static scoping in JavaScript like Dart. It can be observed by the end of the foo1 function. While exiting foo1, the values of x and y are x of foo1_1, y of foo3 respectively. Also, the foo2 and foo3 functions that are at the same scope with the foo1 change the variables of the foo function, not the x of foo1 function. Therefore, there is no dynamic scoping in JavaScript. These scoping rules make the JavaScript language more readable but it decreases the flexibility of the language like in the case of Dart.

The Dart and the JavaScript languages are very similar in the context of scoping rules. The only difference is that in JavaScript if you want to reach the value of a variable before its declaration and assignment, you get undefined as a value. You are not getting an error because the declarations are moved to the top of the scope automatically by language itself. However, in Dart, you can’t reach a variable before its declaration.

Python

Example in Python:

```python
x = "x of global"
y = "y of global"

def foo():
   # print("While entering foo:", x, y)
   # UnboundLocalError: local variable 'x'
   # referenced before assignment
   x = "x of foo"
   y = "y of foo"
   print("In foo:", x + ", " + y)

   def foo1():
       nonlocal x
       # print("While entering foo1:", x, y)
       # UnboundLocalError: local variable 'y'
       # referenced before assignment
       print("\nWhile entering foo1:", x)
       x = "nonlocal x of foo1"
       y = "y of foo1"
       print("In foo1:", x + ", " + y)

       def foo1_1():
           global y
           # print("While entering foo1_1:", x, y)
           # UnboundLocalError: local variable 'x'
           # referenced before assignment
           print("\nWhile entering foo1_1:", y)
           x = "x of foo1_1"
           y = "global y of foo1_1"
           print("While exiting foo1_1:", x + ", " + y)

       foo1_1()
       print("After foo1_1 in foo1:", x + ", " + y)
       foo2()
       print("While exiting foo1:", x + ", " + y)

   def foo2():
       # print("While entering foo2:", x, y)
       # UnboundLocalError: local variable 'x'
       # referenced before assignment
       x = "x of foo2"
       y = "y of foo2"
       print("\nWhile exiting foo2:", x + ", " + y)
       foo3()
```

```
    def foo3():
        print("\nIn foo3:", x + ", " + y)

    foo1()
    print("After foo1 in foo:", x + ", " + y)
    foo2()
    print("While exiting foo:", x + ", " + y)

foo()

print("\nIn global:", x + ", " + y)
```

The output of the example in Python:

```
In foo: x of foo, y of foo

While entering foo1: x of foo
In foo1: nonlocal x of foo1, y of foo1

While entering foo1_1: y of global
While exiting foo1_1: x of foo1_1, global y of foo1_1
After foo1_1 in foo1: nonlocal x of foo1, y of foo1

While exiting foo2: x of foo2, y of foo2

In foo3: nonlocal x of foo1, y of foo
While exiting foo1: nonlocal x of foo1, y of foo1
After foo1 in foo: nonlocal x of foo1, y of foo

While exiting foo2: x of foo2, y of foo2

In foo3: nonlocal x of foo1, y of foo
While exiting foo: nonlocal x of foo1, y of foo

In global: x of global, global y of foo1_1
```

In Python language, the variables can be reached in the inner scopes as can be observed from the “In foo3: nonlocal x of foo1, y of foo” line in the output. However, the variables in the outer scopes can not be changed directly from the inner scopes. When the assignment is tried in the inner scopes, it creates new local variables in the current scope. Because of this reason when I tried to print the x and y variables at the beginning of the functions, I get an error that is “UnboundLocalError: local variable 'x' referenced before assignment”. Therefore, in Python variables cannot be reached before the assignment as the same as the Dart language.

In order to assign a new value to the variables at the outer scopes, there are two reserved words: nonlocal to reach the outer scope but not global scope, global to reach the global scope. In the example, the foo1 function assigns a new value to the x of foo by the nonlocal statement. Also, the foo1_1 function assigns the new value to y in the global scope.

Therefore, the Python language is different from the Dart and JavaScript languages by this feature. These languages let programmers change the variables in the outer scopes directly, but Python forces the programmers to use nonlocal or global words for this purpose. On the other hand, Python prevents unexpected results because of the ability to change the values of the variables in the outer scopes. By this feature, Python decreases the flexibility but increases reliability.

Moreover, in Python language, there is no dynamic scoping. In the example, the foo1 function calls foo2 and then foo2 calls foo3 that are in the same scope with the foo1, but foo3 function prints the values of the x and y in the foo function. Therefore, there is static scoping in Python.

Perl

The code of the static scoping example in Perl:

```
$x = "x of global";
$y = "y of global";

sub foo {
   print "\nWhile entering foo: $x, $y\n";
   $x = "x of foo";
   $y = "y of foo";
   print "In foo: $x, $y\n";

   sub foo1 {
       print "\nWhile entering foo1: $x, $y\n";
       $x = "x of foo1";
       my $y = "my y of foo1";
       print "In foo1: $x, $y\n";

       sub foo1_1 {
           print "\nWhile entering foo1_1: $x, $y\n";
           $x = "x of foo1_1";
           $y = "y of foo1_1";
           print "While exiting foo1_1: $x, $y\n";
       }

       foo1_1();
       print "After foo1_1 in foo1: $x, $y\n";
       foo2();
       print "While exiting foo1: $x, $y\n";
   }

   sub foo2 {
       print "\nWhile entering foo2: $x, $y\n";
       $x = "x of foo2";
```

```
        $y = "y of foo2";
        print "While exiting foo2: $x, $y\n";
        foo3();
    }

    sub foo3 {
        print "\nWhile entering foo3: $x, $y\n";
        $x = "x of foo3";
        $y = "y of foo3";
        print "While exiting foo3: $x, $y\n";
    }

    foo1();
    print "After foo1 in foo: $x, $y\n";
    foo2();
    print "While exiting the foo: $x, $y\n";
}

foo();

print "\nIn global: $x, $y\n";
```

The output of the static scoping example in Perl:

```
While entering foo: , y of global
In foo: x of foo, y of foo

While entering foo1: x of foo, y of foo
In foo1: x of foo1, my y of foo1

While entering foo1_1: x of foo1, my y of foo1
While exiting foo1_1: x of foo1_1, y of foo1_1
After foo1_1 in foo1: x of foo1_1, y of foo1_1

While entering foo2: x of foo1_1, y of foo
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting foo1: x of foo3, y of foo1_1
After foo1 in foo: x of foo3, y of foo3

While entering foo2: x of foo3, y of foo3
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
```

```
While exiting the foo: x of foo3, y of foo3

In global: x of foo3, y of foo3
```

The code and the output of the dynamic scoping example in Perl:

```
#$x = "x of global";
$y = "y of global";

sub foo {
   print "\nWhile entering foo: $x, $y\n";
   $x = "x of foo";
   $y = "y of foo";
   print "In foo: $x, $y\n";

   sub foo1 {
       print "\nWhile entering foo1: $x, $y\n";
       $x = "x of foo1";
       local $y = "my y of foo1";
       print "In foo1: $x, $y\n";

       sub foo1_1 {
           print "\nWhile entering foo1_1: $x, $y\n";
           $x = "x of foo1_1";
           $y = "y of foo1_1";
           print "While exiting foo1_1: $x, $y\n";
       }

       foo1_1();
       print "After foo1_1 in foo1: $x, $y\n";
       foo2();
       print "While exiting foo1: $x, $y\n";
   }

   sub foo2 {
       print "\nWhile entering foo2: $x, $y\n";
       $x = "x of foo2";
       $y = "y of foo2";
       print "While exiting foo2: $x, $y\n";
       foo3();
   }

   sub foo3 {
       print "\nWhile entering foo3: $x, $y\n";
       $x = "x of foo3";
       $y = "y of foo3";
       print "While exiting foo3: $x, $y\n";
```

```
    }

    foo1();
    print "After foo1 in foo: $x, $y\n";
    foo2();
    print "While exiting the foo: $x, $y\n";
}

foo();

print "\nIn global, $x, $y\n";
```

The output of the dynamic scoping example in Perl:

```
While entering foo: , y of global
In foo: x of foo, y of foo

While entering foo1: x of foo, y of foo
In foo1: x of foo1, my y of foo1

While entering foo1_1: x of foo1, my y of foo1
While exiting foo1_1: x of foo1_1, y of foo1_1
After foo1_1 in foo1: x of foo1_1, y of foo1_1

While entering foo2: x of foo1_1, y of foo1_1
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting foo1: x of foo3, y of foo3
After foo1 in foo: x of foo3, y of foo

While entering foo2: x of foo3, y of foo
While exiting foo2: x of foo2, y of foo2

While entering foo3: x of foo2, y of foo2
While exiting foo3: x of foo3, y of foo3
While exiting the foo: x of foo3, y of foo3

In global, x of foo3, y of foo3
```

In Perl, there are both dynamic and static scoping. The static scoping is specified by using my keyword at the beginning of the declaration or not specifying any word. On the other hand, the dynamic scoping is specified by using the word local at the beginning of the declaration of the variable.

In the static scoping example, if there is not a declaration in the current scope until the statement, the Perl language looks for the outer scopes until the global scope. If there is not any declaration of the variable, then it prints nothing for the variable as can be seen from the first print of the foo function. When there is an assignment without my and local keywords and the variable is not declared in the outer scopes, then the Perl declares the variable in the global scope. It is very similar to the older versions of JavaScript. Declaration of the variable with my keyword is that the variable is declared in the local scope even if there is a variable declaration with the same name in the outer scopes. The variables at the outer scopes with the same name are hidden for the inner scope. When we look at the “While entering foo2: x of foo1_1, y of foo” line at the output, we can see that foo1_1 function changes the value of the x of foo but not y of foo.

In the dynamic scoping example, because of the local declaration of the variable y in foo1, when the foo1 calls foo2 and then foo2 calls foo3 function, the foo2 and foo3 changes the y of foo1 even if all the foo1, foo2, and the foo3 are at the same scope. We can observe that from the line “After foo1 in foo: x of foo3, y of foo” in the output. The foo2 and foo3 change the value of the y of foo1, but the x of foo function due to the static scoping of x. This dynamic scoping feature of Perl is not included in the Dart, JavaScript, Python, and PHP.

These scoping rules in Perl increase the flexibility of the language but decrease the readability.

PHP

Example in PHP:

```
<html>
<body>
<?php
$x = "x of global";
$y = "y of global";

function foo() {
   echo "<br>While entering foo: $x, $y<br>";
   $x = "x of foo";
   $y = "y of foo";
   echo "In foo: $x, $y<br>";

   function foo1() {
       echo "<br>While entering foo1: $x, $y<br>";
       $x = "x of foo1";
       global $y;
       $y = "global y of foo1";
       echo "In foo1: $x, $y<br>";

       function foo1_1() {
           echo "<br>While entering foo1_1: $x, $y<br>";
           $x = $GLOBALS['x'];
           $x = "global x of foo1_1";
```

```
            $y = "y of foo1_1";
            echo "While exiting foo1_1: $x, $y<br>";
        }

        foo1_1();
        echo "After foo1_1 in foo1: $x, $y<br>";
        foo2();
        echo "While exiting foo1: $x, $y<br>";
    }

    function foo2() {
        echo "<br>While entering foo2: $x, $y<br>";
        $x = "x of foo2";
        $y = "y of foo2";
        echo "While exiting foo2: $x, $y<br>";
        foo3();
    }

    function foo3() {
        echo "<br>While entering foo3: $x, $y<br>";
        $x = "x of foo3";
        $y = "y of foo3";
        echo "While exiting foo3: $x, $y<br>";
    }

    foo1();
    echo "After foo1 in foo: $x, $y<br>";
    foo2();
    echo "While exiting the foo: $x, $y<br>";
}

foo();

echo "<br>In global: $x, $y<br>";
?>
</body>
</html>
```

The output of the example in PHP:

```
While entering foo: ,
In foo: x of foo, y of foo

While entering foo1: ,
In foo1: x of foo1, global y of foo1

While entering foo1_1: ,
While exiting foo1_1: global x of foo1_1, y of foo1_1
After foo1_1 in foo1: x of foo1, global y of foo1
```

```
While entering foo2: ,
While exiting foo2: x of foo2, y of foo2

While entering foo3: ,
While exiting foo3: x of foo3, y of foo3
While exiting foo1: x of foo1, global y of foo1
After foo1 in foo: x of foo, y of foo

While entering foo2: ,
While exiting foo2: x of foo2, y of foo2

While entering foo3: ,
While exiting foo3: x of foo3, y of foo3
While exiting the foo: x of foo, y of foo

In global: x of global, global y of foo1
```

In PHP, the variables in the outer scopes except global cannot be reached from the inner scopes. By this feature, PHP is very different from other languages. This increases the reliability of the language but decreases flexibility.

In the inner scopes, only the variables in the global scope can be reached by the global statement or global array. In the example, the foo1 function can change the value of the global y by using the global declaration statement. By this statement, y of global is reachable in this scope but not in the inner scopes. In the foo1_1 function, x of global can be changed by declaring x with the globals array.