CS315

Sec01

Homework Assignment 1

Alper Kandemir

21703062

Dart

Theoretical Expectations:

-Inner scope can access the variables declared in the outer scopes

and the variable can be assigned to the new value.

- Inner scopes can access the variables in the closer outer scope.

-The static scoping is used

Experiment:

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

foo (){
        print ("Initially in foo x: " + x);
        //print ("Initially in foo y: " + y);
        //error
        var y = "y in foo";
        x="x in foo";
        print ("Initially in foo y: " + y);
        print ("Initially in foo x: " + x);
        print ("");
        foo1(){

            print ( "In foo1 x: " + x + ", y: " + y);
            x = "x in foo1";
            y = "y in foo1";
            print ( "End foo1 x: " + x + ", y: " + y);
            print ("");
        }

        foo2(){
            print ( "In foo2 x: " + x + ", y: " + y);
            x = "x in foo2";
            y = "y in foo2";
            print ("End foo2 x: " + x + ", y: " + y);
            print ("");
            foo1();

        }

        foo3(){
            print ("In foo3 y: " + y);
            y = "y in foo3";
            print ("In foo3 y: " + y);
            var x = "x in foo3";
            print ("In foo3 x: " + x );
            print ("");
            foo1();
    }
    foo3();
    print ( "After foo3 in foo: " + x + ", y: " + y);
    print ("");
    foo2();
    print( "After foo2 in foo: " + x + ", y: " + y);
    print ("");
    }
print("Before foo in main x: " + x + ", y: " + y);
print ("");
foo();
print("After foo in main x: " + x + ", y: " + y);
}
```

Output:

```
Before foo in main x: main, y: main

Initially in foo x: main
Initially in foo y: y in foo
Initially in foo x: x in foo

In foo3 y: y in foo
In foo3 y: y in foo3
In foo3 x: x in foo3

In fool x: x in foo, y: y in foo3
End fool x: x in fool, y: y in fool

After foo3 in foo: x in fool, y: y in fool

In foo2 x: x in fool, y: y in fool
End foo2 x: x in foo2, y: y in foo2

In fool x: x in foo2, y: y in foo2
End fool x: x in fool, y: y in fool

After foo2 in foo: x in fool, y: y in fool

After foo in main x: x in fool, y: main
```

Result:

Inner scope can access the variables declared in the outer scopes and the variable can be assigned to the new value. Inner scopes can access the variables in the closer outer scope. in foo, x value comes from the outer scope and the new y variable is created. When they are printed inner scope values are printed. also, in foo3, foo1 is called and the value of new x in foo3 can be reached. So, functions can reach the x of the main and y of the foo function or, y of the main and x of the foo3 function. All inner scopes can reach the variables in the outer scopes. Dynamic scope states that the last value assigned to the variable is the value of that variable. Dart uses static scope because there is no behaviour of dynamic scoping in this example.

Javascript

```
<html>
<head>
</head>
<body>
<script>
x = "x of global";
y = "y of global";
function foo () {
    alert( " \n foo x: " + x + ", y: " + y );
    x = "x in foo" ;
    var y = "y in foo" ;
    alert ( "In foo x: " + x + ", y: " + y );

    function foo1 () {
        var x ="x in foo1";
        y = "y in foo1" ;
        alert ( "In foo1: " + x + ", " + y );
        function foo2 () {
            y = "y in foo2" ;
            alert( "In foo2 x: " + x + ", y: " + y );
        }
        foo2 ();
        alert( "End foo2 in foo1 x: " + x + ", y: " + y );
        foo3 ();
        alert( "End foo3 in foo1 x: " + x + ", y: " + y );
    }
    function foo3 () {

        y = "y in foo3" ;
        alert( "In foo3 x: " + x + ", y: " + y );

    }

    foo1 ();
    alert( "End foo1 in foo x: " + x + ", y: " + y );
    foo3 ();
    alert( "End foo3 in foo x: " + x + ", y: " + y );
}
foo ();
alert( "In global x: " + x + ",y: " + y );

{
    let z = "z block scope" ; //block scope
}
// alert ( " In global z: "+ z ); error

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

Output:

```
foo x: x of global, y: undefined
In foo x: x in foo, y: y in foo
In foo1: x in foo1, y in foo1
In foo2 x: x in foo1, y: y in foo2
End foo2 in foo1 x: x in foo1, y: y in foo2
In foo3 x: x in foo, y: y in foo3
End foo3 in foo1 x: x in foo1, y: y in foo3
End foo1 in foo x: x in foo, y: y in foo3
In foo3 x: x in foo, y: y in foo3
End foo3 in foo x: x in foo, y: y in foo3
In global x: x in foo,y: y of global
```

Results:

When the functions declared function creates a new scope. Variables defined in the inner scope are not accessible from the outer scope. a global variable can be accessed from any part. if there is no new declaration, the outer scope variables can be reached and changed from the inner scopes as in Dart. We can see it from the example where the undefined is printed. Javascript has static scope. it can be observed by the foo1() function. Let the keyword indicates the block scope. After the block, the variable cannot be reached like the end of the example in the comment.

Perl

Theoretical Expectations:

- Dynamic and static scopes

Experiment:

Static Scoping:

```
$x = "global" ;
$y = "global" ;
sub foo {
    print "Initially in foo x: ", $x ," y: ", $y ;
    print"\n";

    $x = "x in foo" ;
    $y = "y in foo" ;
    print "In foo x: ", $x ," y: ", $y  ;
    print"\n";
    print"\n";
    sub foo1 {
        print "In foo1 x: ", $x , $y  ;
        print"\n";
        $x = "x in foo1" ;
        my $y = "static y in foo1" ;
        print "In foo1 x: ", $x ," y: ", $y ;
        print"\n";
        print"\n";
        sub foo2 {
            print "In foo2 x: ", $x ," y: ", $y ;
            print"\n";
            $x = "x in foo2" ;
            $y = "y in foo2" ;
            print "exit foo2 x: ", $x ," y: ", $y  ;
            print"\n";
            print"\n";
        }
        foo2();
        print "End foo2 in foo1 x: ", $x ," y: ", $y ;
        print"\n";
        foo3();
        print "End foo3 x: ", $x ," y: ", $y  ;
        print"\n";
        print"\n";
    }
    sub foo3 {
        print "In foo3 x: ", $x ," y: ", $y  ;
        print"\n";
        $x = "x in foo3" ;
        $y = "y in foo3" ;
        print "End foo3 x: ", $x ," y: ", $y  ;
        print"\n";
        print"\n";
    }
    foo1();
    print "End foo1 x: ",  $x ," y: ", $y  ;
    print"\n";
    foo3();
    print "End foo3 x: ", $x ," y: ", $y  ;
    print"\n";
    print"\n";
}
foo();
print "Global x: ", $x ,"y: ", $y ;
print"\n";
```

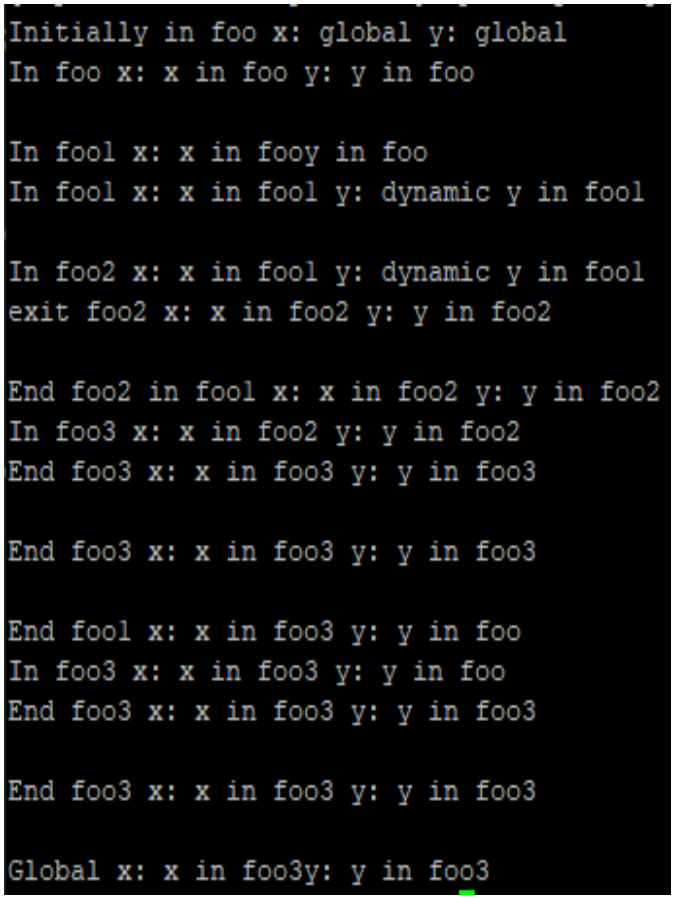
Static Scoping Output:

```
Initially in foo x: global y: global
In foo x: x in foo y: y in foo

In fool x: x in fooy in foo
In fool x: x in fool y: static y in fool

In foo2 x: x in fool y: static y in fool
exit foo2 x: x in foo2 y: y in foo2

End foo2 in fool x: x in foo2 y: y in foo2
In foo3 x: x in foo2 y: y in foo
End foo3 x: x in foo3 y: y in foo3

End foo3 x: x in foo3 y: y in foo2

End fool x: x in foo3 y: y in foo3
In foo3 x: x in foo3 y: y in foo3
End foo3 x: x in foo3 y: y in foo3

End foo3 x: x in foo3 y: y in foo3

Global x: x in foo3y: y in foo3
```

Dynamic Scoping:

```
$x = "global" ;
$y = "global" ;
sub foo {
    print "Initially in foo x: ", $x ," y: ", $y ;
    print"\n";

    $x = "x in foo" ;
    $y = "y in foo" ;
    print "In foo x: ", $x ," y: ", $y  ;
    print"\n";
    print"\n";
    sub foo1 {
        print "In foo1 x: ", $x , $y  ;
        print"\n";
        $x = "x in foo1" ;
        local $y = "dynamic y in foo1" ;
        print "In foo1 x: ", $x ," y: ", $y ;
        print"\n";
        print"\n";
        sub foo2 {
            print "In foo2 x: ", $x ," y: ", $y ;
            print"\n";
            $x = "x in foo2" ;
            $y = "y in foo2" ;
            print "exit foo2 x: ", $x ," y: ", $y  ;
            print"\n";
            print"\n";
        }
        foo2();
        print "End foo2 in foo1 x: ", $x ," y: ", $y ;
        print"\n";
        foo3();
        print "End foo3 x: ", $x ," y: ", $y  ;
        print"\n";
        print"\n";
    }
    sub foo3 {
        print "In foo3 x: ", $x ," y: ", $y  ;
        print"\n";
        $x = "x in foo3" ;
        $y = "y in foo3" ;
        print "End foo3 x: ", $x ," y: ", $y  ;
        print"\n";
        print"\n";
    }
    foo1();
    print "End foo1 x: ",  $x ," y: ", $y  ;
    print"\n";
    foo3();
    print "End foo3 x: ", $x ," y: ", $y  ;
    print"\n";
    print"\n";
}
foo();
print "Global x: ", $x ,"y: ", $y ;
print"\n";
```

Dynamic Scoping Output:



Results:

Perl has dynamic and static scopes. My keyword is used for the static scoping at the beginning of the declaration, and the local keyword is used for the dynamic scoping at the beginning of the declaration of the variable. The assignment without my and local keywords and outer scopes have no declaration of the variable then the global variable is created. In the static scoping example, x and y values are searched at the outer scope. Also, inner scope assignments changes can be reached from outer scope. Foo3() function can access the change made in foo2() unlike the static scope example. Also, foo1() can access the foo3() changes. The Dart, JavaScript, Python, and PHP do not have the dynamic scoping feature of Perl.

PHP

```php
<!DOCTYPE html>
<html>
<body>

<?php
$x = "global" ;
$y = "global" ;
function foo () {
    echo "In foo x: " .$x . " y: ". $y. "<br>"  ;
    $x = "x in foo" ;
    $y = "y in foo" ;
    echo "In foo x: " .$x . " y: ". $y. "<br>"  ;
    echo"<br>";
    function foo1 () {
        $x = "x in foo1" ;
        global $y ;
        $y = "y in foo1" ;
        echo "In foo1 x: " .$x . " y: ". $y. "<br>"  ;
        echo"<br>";
        function foo2 () {
            $x = "x in foo2" ;
            $y = $GLOBALS [ 'y' ];
            $y = "y in foo2" ;
            echo "End In foo1 x: " .$x . " y: ". $y. "<br>"  ;
        }
        foo2 ();
        echo "End foo2 in foo1 x : ". $x . " y: ". $y. "<br>"  ;
        foo3 ();
        echo "End foo3 in foo1 x : ". $x . " y: ". $y. "<br>"  ;
        echo"<br>";
    }
    function foo3 () {
        $x = "x of foo3" ;
        echo "in foo3 x:  ". $x . " y: ". $y. "<br>"  ;
        echo"<br>";

    }
    foo1 ();
    echo "End foo1 in foo x:  ". $x . " y: ". $y. "<br>"  ;
    foo3 ();
    echo "End foo3 in foo x:  ". $x . " y: ". $y. "<br>"  ;
    echo"<br>";
}
foo ();
echo "End foo in global x:  ". $x . " y: ". $y. "<br>"  ;
?>

</body>
</html>
```

Output:

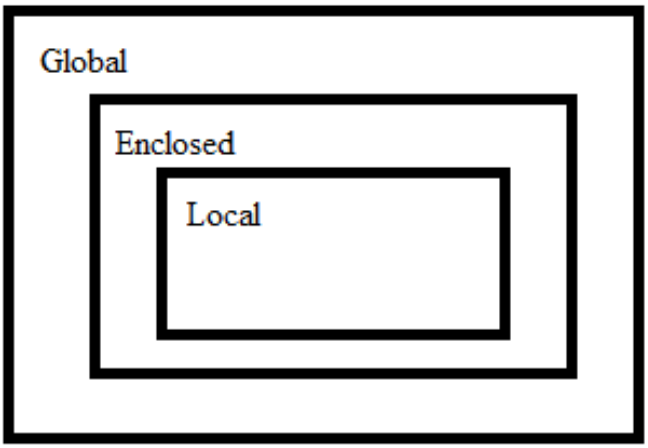
```
In foo x: y: In foo x: x in foo y: y in foo

In foo1 x: x in foo1 y: y in foo1

End In foo1 x: x in foo2 y: y in foo2
End foo2 in foo1 x : x in foo1 y: y in foo1
in foo3 x: x of foo3 y:

End foo3 in foo1 x : x in foo1 y: y in foo1

End foo1 in foo x: x in foo y: y in foo
in foo3 x: x of foo3 y:

End foo3 in foo x: x in foo y: y in foo

End foo in global x: global y: y in foo1
```

Results:

Inner scope cannot reach the global variables. The first print line gives undefined variable error on Dijkstra but the browser will print the x and y without their values. If the global key word is used, global variables can be reachable from inner scope but only for that scope. End of the foo3() function we can see that local y value is used instead of global although global value of y is used in the inner scope. It is increasing the reliability compared to other languages but decreasing the flexibility.

Python

Global, Local and Enclosed scopes will be checked in this test. Local is defined inside function/class, enclosed is defined inside enclosing functions and global is defined at the uppermost level. Nonlocal and global reserved keywords are used to assign a new value to the variables at the outer scopes. Nonlocal reaches the outer scope but it is enclosed in the function, and global reaches the global scope.



Theoretical Expectations:

-The function will check a variable name in its local scope. If it does not find it, the outer scopes will be checked.

-Changes in variables in inner scopes don't change the variable in outer scopes.

-Before the assignment python variables cannot be reached.

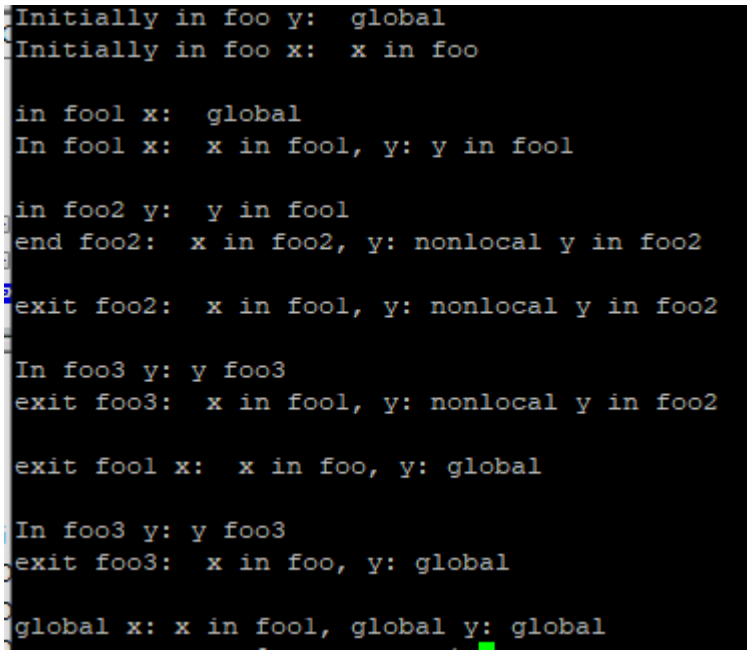
- Nonlocal and global reserved keywords behaves as mentioned above.

-Has only static scoping, no dynamic scoping.

Experiment:

```
x = "global"
y = "global"
def foo ():
    #print ("Initially in foo x: " , x)
    #error because x cannot be used before initialization
    print ("Initially in foo y: " , y)
    #y value is global value. No y variable in this block
    x="x in foo"
    print ("Initially in foo x: " , x)
    def foo1 ():
        global x
        # x is global x
        print ()
        print ("in foo1 x: ", x)
        x = "x in foo1"
        y = "y in foo1"
        print ( "In foo1 x: " , x + ", y: " + y)

        def foo2 ():
            nonlocal y
            #y will be searched at outer scope
            print ()
            print ( "in foo2 y: " , y)
            x = "x in foo2"
            y = "nonlocal y in foo2"
            print ("end foo2: " , x + ", y: " + y)

        foo2()
        print ()
        print ("exit foo2: " , x + ", y: " + y)
        foo3()
        print ("exit foo3: " , x + ", y: " + y)
        #y value is global value
    def foo3 ():

        y = "y foo3"
        print ()
        print ("In foo3 y: "  + y)

    foo1()
    print ()
    print ("exit foo1 x: " , x + ", y: " + y)
    #y value is global value. No y variable in this block
    foo3()
    print ("exit foo3: " , x + ", y: " + y)
    #y value is global value. No y variable in this block
foo()
print ()
print ("global x:" , x + ", global y: " + y)
# global x is changed in the function because of global keyword
#{
#x="block x"
#print (" x:" , x)
#}
#syntax error
```

Output:



Result:

In the foo() function, print ("Initially in foo y: " , y) shows that the function will check a variable name in its local scope. If it does not find it, the outer scopes will be checked because it prints global value of y. Changes in variables in inner scopes don't change the variable in outer scopes. Also, the outer scopes variable is not changed from the inner scope variable with the same name because the new local variable is created. If the variable is called before the assignment of the variable, the error will be occurring like in the first comment in foo function. The Dart language has the same behaviour. When the nonlocal keyword is used for y, value of y is searched at outer scope of that function, and y value will be the y value in foo1 at foo2's first print statement. Global keyword is changed the global x, and when the first print statement is called after global keyword, global value of x is printed. This feature differentiates the python from Dart and JavaScript. So, python prevents unexpected results however it decreases flexibility but increases reliability. Dynamic scope states that the last value assigned to the variable is the value of that variable. Python uses static scope because there is no behaviour of dynamic scoping in this example.

Strategies And Conclusion:

First of all, I searched the scope rules of each language on google. In addition, I tried to look at known and reliable sources. Then I listed the rules after reviewing the examples on the sites. After completing my preliminary work, I prepared an example to show all the rules. Then I tested my examples using the online compiler. Interestingly, using an online compiler is a bit risky because sometimes it can give different results, and the results can change from one site to another. In addition, the examples given by the teacher helped me to create my own examples. I also used the book for some parts. I wrote my test codes using VS code, and then I tested them on Dijkstra. PHP and JavaScript experiments were a bit difficult because of their HTML codes. I tested the JavaScript codes using the HTML codes, but when doing the same for PHP, I got a complex output. So, I checked the result using the online compiler. I asked questions to some of my friends. These are mostly the errors I got, and it was related to HTML because sometimes, no matter how hard we look, we can't see the cause of the error. Also, I didn't have any difficulties in my experiments except for Dart because of the scoping subject in the Exam, but then I realized that Dart is very similar to other languages that use static scope.

As an online compiler, I liked w3schools the most because it gave more accurate results than others. but it is not very easy to use and also a bit slow. I like to write code using Vs Code, but an online compiler is also good enough. The only problem is having to open multiple tabs for each language.

I used FileZilla and putty to run my code on Dijkstra. To me, this is the easiest way to run the codes via Dijkstra. I checked it by running my codes through putty by sending the codes into my own account one by one.

When looking at languages, the methods they apply on scope increase the flexibility of the language, while reducing readability. On the contrary, as readability increases, flexibility decreases. In other words, while increasing one thing, it is necessary to give up another feature. Dart language and JavaScript are similar in scope rules. Besides, using a static scope is more advantageous than using a dynamic scope. Only Perl uses dynamic scope, and with dynamic scope, the disadvantages outweigh the advantages. According to this experiment, if we had to choose the best language in terms of readability, flexibility and reliability, it would be python. Because outer variables cannot be changed from the inside function. This increases reliability also, nonlocal and global keywords increase the flexibility

because It offers a choice of usage according to the needs, and readability is increased by meaningful keywords, unlike Perl.