21903419
Berk Saltuk Yılmaz

CS315 **Homework 1 - Report**

Scoping in Different Languages

1. Perl

Introduction

Perl supports both dynamic and static scoping. Additionally, it is possible to create a scope for a variable using blocks and packages, and these are called block scope and package scope, respectively.

Inspection of Code

```
$x = "Mr. Worldwide";

sub localCeleb1(){
    $x = "Mr. Localcelebrity"; # this does not create a local scope
    print "In localCeleb1, x is : $x\n";
}

print "Before calling localCeleb1, x is : $x\n";
localCeleb1();
print "After calling localCeleb1, x is : $x\n";
>> Output:
>> Before calling localCeleb1, x is : Mr. Worldwide
>> In localCeleb1, x is : Mr. Localcelebrity
>> After calling localCeleb1, x is : Mr. Localcelebrity
```

In Perl, when one directly assigns a value to a variable that has the same name as in the global scope without specifying with keywords my or local, the variable is named x in this case. This variable's scope is not restricted to the function's scope, and it changes the value of the global variable.
To have a variable whose scope is restricted by the function, the following works:

```
$y = "Mr. Worldwide";

sub localCeleb2(){
    my $y = "Mr. Localcelebrity"; # this creates a local scope
    print "In localCeleb2, y is : $y\n";
}
print "Before calling localCeleb2, y is : $y\n";
localCeleb2();
print "After calling localCeleb2, y is : $y\n";
>> Output:
>> Before calling localCeleb2, y is : Mr. Worldwide
>> In localCeleb2, y is : Mr. Localcelebrity
>> After calling localCeleb2, y is : Mr. Worldwide
```

When the variable is initialized with my or local keyword, this value becomes local to the function and does not affect the value of the global variable.

```
$a = "global a";

sub foo(){
    my $a = "foo a"; # this is only visible in scope of foo, static scoping
    local $b = "foo b";
    $c = "I am c!"; # This is global declaration

    sub boo(){
        local $b = "I am b!"; # this creates dynamic scoping
        loo();
    }

    sub loo(){
        print "In loo, a : $a, b : $b, c : $c\n";
    }

    boo();
    loo();
}

print "In main before foo, a : $a, b : $b, c : $c\n";
foo();
print "In main after foo, a : $a, b : $b, c : $c\n";
loo();
>> Output:
>> In main before foo, a : global a, b : , c :
>> In loo, a : foo a, b : I am b!, c : I am c!
>> In loo, a : foo a, b : foo b, c : I am c!
>> In main after foo, a : global a, b : , c : I am c!
>> In loo, a : foo a, b : , c : I am c!
```

Perl has both dynamic and static scoping, as I mentioned in the introduction part. First of all, when a variable is initialized without my or local keyword, it becomes a global variable independent of where it was declared. In the above example, the variable a defined outside is global. Also, the variable c was declared in the foo function becomes a global variable. When a variable is declared with my keyword, this variable is statically scoped. For instance, the variable a defined in foo will have a static scope, and the functions loo and foo will look for variable a in their static parents, and variable a will have the value "foo a" only in the foo function. On the other hand, variable b will have the dynamic scope, and its value will be passed to the dynamic children since it was declared with the local keyword. Therefore, the loo function will print the value of b as "foo b" when called from the foo function, and it will print "I am b!" when it is called from the boo function. Moreover, since only variables a and c have global declarations, only these variables will be visible after the execution of foo. Lastly, as seen in the example, calling a nested function from the main is valid.

```
$username = "berksaltuk";
{
    my $username = "bsy"; # this creates a block scope
    print "Username is $username\n";
}
print "Username is $username\n";
>> Output:
>> Username is bsy
>> Username is berksaltuk
```

Perl also has block scoping, and it has the same logic as function scope; when a variable is created with my or local keyword, it will be local to the block that it was declared. In the above example, if we did not use my keyword, it would overwrite the global value of the username.

```
my @array = (1..5);
$loopVar = "I feel normal";
print("$loopVar\n");
for my $index (@array){
    # If we don't use my keyword the value of outer loopVar changes
    $loopVar = "I feel like doing the same thing again and again";
    print("$loopVar\n");
}
print("$loopVar\n");
print("$index"); # This will not be printed since within for loop
>> Output:
>> I feel normal
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
```

Loops have the same logic as block scoping, and indeed this is also block scoping. The value of loopVar will be overwritten in the for a loop since it has no my or local keyword in the loop. The critical point is that the variable index is for loop specific since it was declared with my keyword for loop initialization.

```
$kitten = "Meow";

print "Kitten says $kitten\n";

package berkPackage;

    print "Kitten says $kitten\n";
    $kitten = "Roar";

    print "Kitten says $kitten\n";

    print "Kitten in main package says $main::kitten\n";

    our $dog = "What da dog doin?"; #Other packages can access this
```

```
package jaxPackage; # Jax is one of my dogs
    print "They always ask $dog, but they never ask how da dog doin? :(\n";
>> Output:
>> Kitten says Meow
>> Kitten says
>> Kitten says Roar
>> Kitten in main package says Meow
>> They always ask What da dog doin?, but they never ask how da dog doin? :(
```

As the only language with dynamic scoping in this homework, Perl also has an interesting scoping rule: package scoping. When you declare a package, the variables defined after this declaration become local to this package, independent of my or local keywords. If one wants to access the main package from a user-defined package, main:: <variable_name> must be used; this is similar to C++'s namespaces. Moreover, if you want a variable to be visible from other packages, this variable must be declared with our keyword, in the above example, the dog variable is accessible from both packages since it was declared with our keyword. Note that this feature only exists in Perl. Therefore, this example was not included in the code of other languages.

2. Python

Introduction

Python only supports static scoping. It does not have a block scope and package scope; however, it is possible to access variables in different scopes with keywords global and nonlocal. Moreover, function definitions in Python can be used to simulate block scope.

Inspection of Code

```
x = "Mr. Worldwide";
def localCeleb1():
    global x # In python to change the global value, we must declare variable as global
    x = "Mr. Localcelebrity" # it creates local variable otherwise unlike Perl
    print("In localCeleb1 x is : " + x)
print("Before calling localCeleb1 x is : " + x)
localCeleb1()
print("After calling localCeleb1 x is : " + x)
>> Output:
>> Before calling localCeleb1 x is : Mr. Worldwide
>> In localCeleb1 x is : Mr. Localcelebrity
>> After calling localCeleb1 x is : Mr. Localcelebrity
```

In Python, if we assign a value to a variable, even if there is a variable in the main with the same name, Python creates a local variable and assigns the value to this new variable. To access a global variable, you should call this variable with the global keyword.

```
y = "Mr. Worldwide"

def localCeleb2():
    y = "Mr. Localcelebrity"
    print("in localCeleb2 is y : " + y)
print("Before calling localCeleb2, y is : " + y)
localCeleb2()
print("After calling localCeleb2, y is : " + y)
>> Output:
>> Before calling localCeleb2, y is : Mr. Worldwide
>> in localCeleb2 is y : Mr. Localcelebrity
>> After calling localCeleb2, y is : Mr. Worldwide
```

As seen in the above example, if we do not specify this variable with the global keyword, Python only looks for this variable within the scope of the function and creates a local variable since there is no variable named y in the function's scope. Moreover, if we would directly print y without assigning a value to it, it would print "Mr. Worldwide," which is the value given in the localCeleb2 function's static parent.

```
a = "global a"
b = "" # I have added these two lines unlike perl because
c = "" # python does not print empty strings for undefined values
def foo():
    a = "foo a"
    b = "foo b"
    c = "I am c"

    def boo():
        # b = "I am b!" this will not have an effect as it did in Perl
        nonlocal b
        b = "I am b!" # This changes the value of b in its static parent
        loo()

    def loo():
        print("In loo, a : " + a + ", b : " + b + ", c : " + c)
    boo()
    loo()

print("In main before foo , a : " + a + ", b : " + b + ", c : " + c)
foo()
print("In main after foo, a : " + a + ", b : " + b + ", c : " + c)
// loo() // Error: loo() is not defined
>> Output:
>> In main before foo , a : global a, b : , c :
>> In loo, a : foo a, b : I am b!, c : I am c
>> In loo, a : foo a, b : I am b!, c : I am c
>> In main after foo, a : global a, b : , c :
```

In the above example, different from Perl, the variables defined without a keyword do not become global. Moreover, Python only has static scoping, and to change the value of a variable given in the static parent, we should use the nonlocal keyword. In the above example, in the function boo, we change the value of b in the function foo using the nonlocal keyword. If we did not use this keyword, Python would create a local variable. Moreover, loo looks for static parents and prints the values independent from which function is called it. Lastly, unlike Perl, in Python, it is not possible to call a nested function from outside.

```
username = "berksaltuk"
def block():
    username = "bsy"
    print("Username is " + username)
block()
print("Username is " + username)
>> Output:
>> Username is bsy
>> Username is berksaltuk
```

In Python, it is not possible to create blocks since the language was built on indentations; however, it is possible to simulate block scoping to some extent by using function definitions. In the above example, the result is the same with Perl when we create a block and declare a variable with my keyword.

```
loopVar = "I feel normal"
print(loopVar)
for i in range(5):
    loopVar = "I feel like doing the same thing again and again"
    # for loop does not create the effect of block scoping
    print(loopVar)
print(loopVar)
print(i) # Unlike Perl variable i is reachable outside
>> Output:
>> I feel normal
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> I feel like doing the same thing again and again
>> 4
```

In Python, inside the for loop when we change the value of a variable defined outside, Python does not create a local variable but changes the value of the outside variable unlike it does in functions. Moreover, it is possible to access variable i even if the for loop is finished, recall that it was not possible in Perl.

3. JavaScript

Introduction

Like Python, JavaScript only supports static scoping. Moreover, it is possible to have block scope using the let keyword. It has global variables accessible from every function or block.

Inspection of Code

```
var x = "Mr. Worldwide";
function localCeleb1(){
    x = "Mr. Localcelebrity"; // Defining a variable without var or let keyword makes it
global
    console.log(`In localCeleb1, x is : ${x}`);
}
console.log("Before calling localCeleb1, x is : " + x);
localCeleb1();
console.log("After calling localCeleb1, x is : " + x);
>> Output:
>> Before calling localCeleb1, x is : Mr. Worldwide
>> In localCeleb1, x is : Mr. Localcelebrity
>> After calling localCeleb1, x is : Mr. Localcelebrity
```

In JavaScript, like in Perl, when a value is assigned to a variable without specifying let or var keywords it is treated as global, and the value of the global variable changes. JavaScript does not create a local variable as in Python.

```
var y = "Mr. Worldwide";
function localCeleb2(){
    var y = "Mr. Localcelebrity"; // y is defined only in localCeleb2 function
    console.log("In localCeleb2, y is : " + y);
}
console.log("Before calling localCeleb2, y is : " + y);
localCeleb2();
console.log("After calling localCeleb2, y is : " + y);
>> Output:
>> Before calling localCeleb2, y is : Mr. Worldwide
>> In localCeleb2, y is : Mr. Localcelebrity
>> After calling localCeleb2, y is : Mr. Worldwide
```

However, in the above example, when the variable y is created with var keyword, it becomes a local variable of the function, hence does not change the value of global y.

```
a = "global a";
b = "global b";
c = "global c";
function foo(){
    var b = "Foo b"
    function boo(){
        var b = "I am boo b!"; // No dynamic scoping
        c = "I am boo c!";
        loo();
    }
    function loo(){
        let c = "Loo c";
        console.log("In loo, a : " + a + ", b : " + b + ", c : " + c);
    }
    boo();
    loo();
}
console.log("In main before foo, a : " + a + ", b : " + b + ", c : " + c);
foo();
console.log("In main after foo, a : " + a + ", b : " + b + ", c : " + c);
//loo(); // Error! loo is not defined!
>> Output:
>> In main before foo, a : global a, b : global b, c : global c
>> In loo, a : global a, b : Foo b, c : Loo c
>> In loo, a : global a, b : Foo b, c : Loo c
>> In main after foo, a : global a, b : global b, c : I am boo c!
```

JavaScript also has only static scoping. I want to address that when a variable was not defined in Perl it does not give an error but prints an empty string for these variables. However, other languages give errors. Therefore, in the above example, all variables are defined as global variables. As seen in the example, function loo looks for the static parent while printing the values, it is independent of the caller function. Moreover, variable c in the boo function becomes global since it was declared without a keyword in front of it. Additionally, after the execution of foo, only global versions of the variables are printed, and note that the value of c was changed. Lastly, as in Python, JavaScript raises an error if we try to call a nested function from outside.

```
var berk = "I hate being in blocks!";
var tiny = "I'd like to have a juicy steak!";
console.log("In main before block berk says: " + berk + ", tiny says: " + tiny);
{
    var berk = "Who put me in a block? I said that I hate it!";
    let tiny = "Please take me out!"; // let keyword creates block scope
    console.log("In block berk says: " + berk + ", tiny says: " + tiny);
}
console.log("In main after block berk says: " + berk + ", tiny says: " + tiny);
>> Output:
>> In main before block berk says: I hate being in blocks!, tiny says: I'd like to have
a juicy steak!
>> In block berk says: Who put me in a block? I said that I hate it!, tiny says: Please
take me out!
>> In main after block berk says: Who put me in a block? I said that I hate it!, tiny
says: I'd like to have a juicy steak!
```

An interesting behavior of JavaScript faces us in block scoping, when a variable is created with the var keyword it does not become local to the block; however, when we initialize the variable with the let keyword, the variable becomes local to the block. The above example shows that behavior clearly.

```
var loopVar = "I feel normal";
var loopVar2 = "I do not feel normal";
console.log("In main #1, loopVar says: " + loopVar + ", loopVar2 says : " +loopVar2);
for( let i = 0; i < 5; i++)
{
    let loopVar = "I feel like in a loop";
    var loopVar2 = "I feel normal now";
  console.log("in for loop, loopVar says: " + loopVar + ", loopVar2 says : " +loopVar2);
}
console.log("In main #2, loopVar says: " + loopVar + ", loopVar2 says : " +loopVar2);
// console.log(i) //Error!
>> Output:
>> In main #1, loopVar says: I feel normal, loopVar2 says : I do not feel normal
>> in for loop, loopVar says: I feel like in a loop, loopVar2 says : I feel normal now
>> in for loop, loopVar says: I feel like in a loop, loopVar2 says : I feel normal now
>> in for loop, loopVar says: I feel like in a loop, loopVar2 says : I feel normal now
i>> n for loop, loopVar says: I feel like in a loop, loopVar2 says : I feel normal now
>> in for loop, loopVar says: I feel like in a loop, loopVar2 says : I feel normal now
>> In main #2, loopVar says: I feel normal, loopVar2 says : I feel normal now
```

When it comes to the for loop, the case is the same with block scoping, when we define the variable with the var keyword, it does not become block specific and changes the value of the global variable; however, the let keyword makes the variable block specific. Moreover, when we initialize the variable i of for loop with the let keyword, it is not accessible outside of the for loop but if we define it with var, i would be printed as 5.

4. PHP

Introduction

When it comes to the block statements such as if statements and loops, PHP supports static scoping; however, when it comes to the functions, PHP has function scope, that is, a function can only access its own variables. PHP functions cannot access the variables defined outside the function if one does not define the variable with the global keyword (or $GLOBALS array). In PHP it is possible to create blocks but it does not have block scope. It also does not have a package scope feature. It also has static variables that can be accessible by initializing only once.

Inspection of Code

```
$x = "Mr. Worldwide";

function localCeleb1(){
    $x = "Mr. LocalCelebrity"; // This is only visible in localCeleb1
    echo "In localCeleb1, x is : ".$x."<br>";
}

echo "Before calling localCeleb1, x is : ".$x."<br>";
localCeleb1();
echo "After calling localCeleb1, x is : ".$x."<br>";
// $y = "Mr.Worldwide";
// function localCeleb2(){
//     echo "In localCeleb1, y is : ".$y."<br>";
// }
// localCeleb2();
// If we directly try to call a global variable
// inside a function we will have an error saying
// y is undefined, you can uncomment and try
>> Output:
>> Before calling localCeleb1, x is : Mr. Worldwide
>> In localCeleb1, x is : Mr. LocalCelebrity
>> After calling localCeleb1, x is : Mr. Worldwide
```

In PHP, I can say that there is a strict function scoping. When we assign a value to a variable, if it is not defined in that function PHP creates a local variable and assigns the value. However, different from other languages, when we directly try to print the value of a global variable inside a function, PHP raises an error saying the variable is not defined.

```php
$y = "Mr. Worldwide";

function localCeleb2(){
    global $y;
    $y = "Mr. Localcelebrity"; // this directly changes global y
    // or $GLOBALS['y'] = "Mr. Localcelebrity";
    echo "In localCeleb2, y is : ".$y."<br>";
}

echo "Before calling localCeleb2, y is : ".$y."<br>";
localCeleb2();
echo "After calling localCeleb2, y is : ".$y."<br>";
>> Output:
>> Before calling localCeleb2, y is : Mr. Worldwide
>> In localCeleb2, y is : Mr. Localcelebrity
>> After calling localCeleb2, y is : Mr. Localcelebrity
```

Therefore, even for printing a global variable we must use the global keyword in front of the variable, or we must use the special array called $GLOBALS. In the above example, we can access and change the value of global y.

```php
$a = "global a";
$b = "";
$c = "";
function foo(){
    $a = "foo a";
    $b = "foo b";
    $c = "I am c!";

    function boo(){
        $b = "I am b!";
    }
    if (1==1){

        echo "In if statement, a : ".$a.", b : ".$b.", c : ".$c."<br>";
    }
    boo();
}
echo "In main before foo, a : ".$a.", b : ".$b.", c : ".$c."<br>";
foo();
echo "In main after foo, a : ".$a.", b : ".$b.", c : ".$c."<br>";
boo(); // This is not problematic
>> Output:
>> In main before foo, a : global a, b : , c :
>> In if statement, a : foo a, b : foo b, c : I am c!
>> In main after foo, a : global a, b : , c :
```

Because of the strict function scoping, I have changed the function loo() example with an if statement in the PHP example. Functions cannot see the variables defined in static parents when it comes to the functions; however, block statements are printing the values defined in static parents in PHP. Moreover, as in Perl, in PHP it is possible to call nested functions from the main program, it does not raise an error and it executes properly.

```
$username = "berksaltuk";
{
    $username = "bsy"; // This directly changes the variable declared outside of the
block
    echo "Username is ".$username."<br>";
}
echo "Username is ".$username."<br>";
>> Output:
>> Username is bsy
>> Username is bsy
```

As I mentioned above, in blocks or block statements, PHP looks for static parents and in the above example, it finds the variable username in the main and changes its value with the given in the block.

```
$loopVar = "I feel normal"; // You can comment out this declaration
echo "$loopVar<br>";
for( $i = 0; $i < 5; $i++)
{
    $loopVar = "I am in a loop now!";
    echo "$loopVar<br>";
}
echo "$loopVar<br>"; // Even if this variable were declared in for it is accessible

echo "$i<br>"; // This is still accessible
>> Output:
>> I feel normal
>> I am in a loop now!
>> I am in a loop now!
>> I am in a loop now!
>> I am in a loop now!
>> I am in a loop now!
>> I am in a loop now!
>> 5
```

Moreover, in loops, as in the block scoping example, the value of the global variable changes from the inside of for loop. Moreover, the variable i given in for initialization is still accessible even if the for loop is finished.

```
function staticFoo(){
    static $waow = 0;
    $zoo = 0;
    $zoo++;
    $waow++;
    echo "zoo is $zoo, waow is $waow<br>";
    // zoo will be 0 everytime but waow will be updated
}
staticFoo();
staticFoo();
>> Output:
>> zoo is 1, waow is 1
>> zoo is 1, waow is 2
```

I wanted to add the static variables in PHP. When a variable is defined as static even if the function removes from the stack after its execution, the static variable remains and keeps its value.

5. Dart

Introduction

Dart language also supports only static scoping. In Dart, one can access global variables from the inside of a function or create local variables with the same name as global variables with the var keyword. Dart also supports block scoping again by using the var keyword. Dart does not support package scope, and also unlike languages such as Python, it is not possible to access a variable that is defined in for loop definition.

Inspection of Code

```
var x = "Mr. Worldwide";

void localCeleb1(){
  x = "Mr. Localcelebrity"; // This sees the variable defined in main and changes it
  print('In localCeleb1, x is : $x');
}

print('Before calling localCeleb1, x is : $x');
localCeleb1();
print('After calling localCeleb1, x is : $x');
>> Output:
>> Before calling localCeleb1, x is : Mr. Worldwide
>> In localCeleb1, x is : Mr. Localcelebrity
>> After calling localCeleb1, x is : Mr. Localcelebrity
```

As in Perl and JavaScript, if we do not specify the variable with a keyword, var in this case, the variable is matched with the global one and Dart changed its value.

```
var y = "Mr. Worldwide";

void localCeleb2(){
  var y = "Mr. Localcelebrity"; // this creates a local variable
  print('In localCeleb1, y is : $y');
}
print('Before calling localCeleb1, y is : $y');
localCeleb2();
print('After calling localCeleb1, y is : $y');
>> Output:
>> Before calling localCeleb1, y is : Mr. Worldwide
>> In localCeleb1, y is : Mr. Localcelebrity
>> After calling localCeleb1, y is : Mr. Worldwide
```

If we define the variable with the var keyword, it becomes a local variable of the function and it is visible in the function's scope.

```
var a = "global a";
var b = ""; // If we do not declare these variables
var c = ""; // compiler gives error
void foo(){
    var a = "foo a";
    var b = "foo b";
    c = "I am c!";
    void loo(){
      print("In loo, a : $a, b: $b, c : $c");
    }
    void boo(){
      var b = "I am b!";
      loo();
    }
    boo();
    loo();
}
print("In main before foo, a : $a, b : $b, c : $c");
foo();
print("In main after foo, a : $a, b : $b, c : $c");
// boo(); // Error, undefined function!
>> Output:
>> In main before foo, a : global a, b : , c :
>> In loo, a : foo a, b: foo b, c : I am c!
>> In loo, a : foo a, b: foo b, c : I am c!
>> In main after foo, a : global a, b : , c : I am c!
```

Again, in Dart too, if we try to print undefined variables, it gives an error. Moreover, as in Perl and JavaScript, c becomes a global variable even if it was declared in the function foo since it was not declared with the keyword var. Moreover, this example shows that Dart is only supporting static scope and the printed values in the callee function boo do not change depending on the caller functions, foo, and boo. Moreover, we can see b is not changed in global since it was defined locally in the functions. Lastly, it is not legal to call a nested function from outside, Dart compiler gives an error saying that the function is not defined.

```
var username = "berksaltuk";
{
  var username = "bsy";
  print("Username is $username"); // Dart supports block scoping
}
print("Username is $username");
>> Output:
Username is bsy
Username is berksaltuk
```

When we define a variable with the var keyword inside a block, this variable becomes a local variable within the scope of this block, so we can say that Dart is also supporting block scope. Note that if the keyword var was not given, the global value would be changed into bsy.

```
var loopVar = "I feel normal";
print(loopVar);
for( var i = 0; i < 5; i++)
{
  // If we add var keyword at the line below, the loopVar would be different from the
one in main
  loopVar = "I feel like I am in a loop!";
  print(loopVar);
}
print(loopVar);
>> Output:
>> I feel normal
>> I feel like I am in a loop!
>> I feel like I am in a loop!
>> I feel like I am in a loop!
>> I feel like I am in a loop!
>> I feel like I am in a loop!
>> I feel like I am in a loop!
```

Again, in the for loop, the outside value of loopVar changes and if we would add var to the definition in the for loop, it would not change. We cannot also access the variable i after the for loop terminates.

6. A Brief Comparison of All Five Languages

As a result, we have observed that the only language that supports dynamic scoping is Perl, it uses my for static scoping and local for dynamic scoping. All other languages support only static scoping. Moreover, Dart, Perl, and JavaScript support block scoping with the help of special keywords: var, my/local, and let respectively. We may say that in all of the languages it is possible to access and change global variables through special keywords. Only Perl has a special kind of scoping which is package scope. For all languages, I can say that we must know the scoping rules in detail for correct results because while we are thinking we have created local variables, we might be changing the global ones or vice versa.

7. My Learning Strategy

Before getting my hands on the homework, I skimmed through our lecture notes. After doing that I have read chapter 5 of our textbook which is about names, bindings, and scopes [1]. After completing this reading, I have executed Altay Güvenir's example codes about scoping on the Dijkstra machine [2]. After completing them, I have started writing the example codes starting with Perl, at the same time I have read the documentation of the language and information written online including GeeksForGeeks and PerlMonks [3], [4], [5]. I have chosen Perl first because it seemed interesting with its different features compared to other languages in this homework. I have written the code in VSCode but I could not execute it there; therefore, I have used TutorialsPoint's online Perl Compiler [6], [7]. I have written different codes first and then I have adapted them to a format that can be writable in other languages, and of course, demonstrate distinguishing features of Perl language. After completing this section, I have started to write Python, I have basically adapted the code I wrote for Perl to Python. This time I did not read documentations since I was familiar with Python's syntax. However, I had quick research about the LEGB (Local, Enclosing, Global, Built-in) rule of Python [8], I directly executed Python from VSCode since I have Python on my local machine. I have mostly played with global and nonlocal keywords while trying python. After getting done with Python, I have written the JavaScript, I also studied JavaScript recently and I am familiar with its syntax; however, I have researched mdn web docs for more information on JavaScript scoping [9]. Again I have used VSCode for writing the programs and running them on my machine since I had Node.js installed on my machine. I have mostly played with let and var keywords while writing the JavaScript program. After completing JavaScript, I have started writing PHP, last semester I took the databases course, I was familiar with PHP, I adapted my code to PHP and run this code using XAMPP [10], I was surprised to see that we cannot reach variables from the inside of a function. Finally, I have started researching about Dart language. I left the Dart to the end because it was the only language I had no idea about. Basically, I have skimmed through its sample programs [11], and I have read the Dart Language Tour's scope part [12]. After completing reading them, I have adapted my programs using Dart's online compiler [13]. Dart language's syntax was easier than I have expected, it was like a hybrid version of several languages. After completing the final piece of the coding part, I have run each program on the Dijkstra machine. After getting the programs run successfully, I have added code snippets and their outputs to my report. After that, I added brief discussions for each code snippet to make them clear. I have also added explanatory comments to the code itself. I believe this report includes brief but helpful information about the scoping of five different languages, and it also shows that languages are not that different when you examine them in a sequential manner.

8. References

[1] Robert W. Sebesta, *Concepts of Programming Languages*. Pearson, 2016.

[2] Altay Güvenir, https://dijkstra.ug.bcc.bilkent.edu.tr/~guvenir/. [Accessed on: April 8, 2022].

[3] "Perl Documentation". https://perldoc.perl.org. [Accessed on: April 9, 2022].

[4] "Perl Scope of Variables", GeeksForGeeks. Available: https://www.geeksforgeeks.org/perl-scope-of-variables/. [Accessed on: April 9, 2022].

[5] "Scoping," PerlMonks. Available: https://www.perlmonks.org/?node_id=66677. [Accessed on: April 9, 2022].

[6] "Visual Studio Code". https://code.visualstudio.com. [Accessed on: April 11, 2022].

[7] "Online Perl Compiler," TutorialsPoint Codingground. https://www.tutorialspoint.com/execute_perl_online.php. [Accessed on: April 9, 2022].

[8] "Python Scope & the LEGB Rule: Resolving Names in Your Code," Real Python. Available: https://realpython.com/python-scope-legb-rule/. [Accessed on: April 9, 2022].

[9] "Scope," mdn web docs. Available: https://developer.mozilla.org/en-US/docs/Glossary/Scope. [Accessed on: April 9, 2022].

[10] "XAMPP". https://www.apachefriends.org/tr/index.html. [Accessed on: April 11, 2022].

[11] "Language Samples," Dart. Available: https://dart.dev/samples. [Accessed on: April 10, 2022].

[12] "A tour of the Dart language," Dart. Available: https://dart.dev/guides/language/language-tour. [Accessed on: April 10, 2022].

[13] "DartPad". https://dartpad.dev/. [Accessed on: April 10, 2022].