CS315

Programming Languages

Fall 2019 Homework 1

Muhammad Arham Khan

21701848

Section 3

29 Nov, 2019

BRIEFING

1. **Main():** the function is called first and shows local scope of x variable and tries accessing the global variable y. Also, this function has a nested function (foo1) defined in it to test scoping rules. Finally the function calls foo2() and foo1() multiple times to display changes in the variables by children scopes.

2. **Foo1():** the function edits the x variable in parent, prints both x (from parent) and y (from global scope) and then edits y in global scope

3. **Foo2():** this function tests the scoping used in the language (dynamic or static) by referencing a variable y from parent scope and printing it along with local x

4. **Foo3():** this function is aimed to check the availability of variables outside control/ loop blocks and so it basically creates a variable in a loop block and tests if it is available outside the block

- **Python:**
  - **CODE:**

```
#global variable declaration
x = "global"
y = "global"

def main():
  x = "main"

  #should output the variable defined in scope of main()
  print("x in main: " + x)
  #should output the y defined in global scope
  print("y in main: " + y)

  def foo1():
    #editing x variable in parent
    nonlocal x
    x = "main->foo1"
```

```
    #y variable is global y
    global y

    print("x in foo1: " + x)
    print("y in foo1: " + y)

    #edits y in parent scope (ie: global)
    y = "global->foo1"

  foo2()
  foo1()

  #should output the edited x defined in scope of main()
  print("x in main2: " + x)

  foo2()
def foo2():
  #since static scoping, global y printed
  #local x printed
  x = "foo2"
  print("x in foo2: " + x)
  print("y in foo2: " + y)

def foo3():
  print("i in loop: ", end="")
  for i in range (3):
    #i is accessible inside block
    a = i
    print( str(i) + ',', end="")
  print()
  #i accessible
  print("i outside loop block in foo3: " + str(i));
  #a accessible outside block
  print("a outside loop block in foo3: " + str(a));
main()
print()
foo3()
```

- **OUTPUT:**

```
x in main: main
y in main: global
x in foo2: foo2
y in foo2: global
x in foo1: main->foo1
y in foo1: global
x in main2: main->foo1
x in foo2: foo2
y in foo2: global->foo1

i in loop: 0,1,2,
i outside loop block in foo3: 2
a outside loop block in foo3: 2
```

- **ANALYSIS:**

Python is a static scoped language that follows the lexical order for scoping variables. So, although it is possible to access the variable in parent scopes using “nonlocal keyword” and the global variable using “global” keywords, it is not possible to readily access the variable in dynamic parent of the current block. Also, once a variable is declared as non-local or global in a block, it is impossible to declare a variable with the same name in the current block as that variable is already bound to a parent/ global scoped variable. So, once declared as non-local/ parent, all edits to variable alter the parent variable as shown in the code output above. Finally, variables declared in control/ loop blocks are available outside the block as well and can be manipulated accordingly. Also, python supports nested scoping and function declaration.

• Dart:

- **CODE:**

```
//global variable declaration
String x = "global";
String y = "global";

void main(){
  String x = "main";
  //should output the variable defined in scope of main()
  print("x in main: " + x);
  //should output the y defined in global scope
  print("y in main: " + y);
```

```
  void foo1(){
    //editing x variable in parent
    x = "main->foo1";

    print("x in foo1: " + x);
    print("y in foo1: " + y);

    //edits y in parent scope (ie: global)
    y = "global->foo1";
  }
  foo2();
  foo1();
  //should output the edited x defined in scope of main()
  print("x in main2: " + x);
  foo2();

  foo3();
}
void foo2(){
  //since static scoping, global y printed
  //local x printed
  String x = "foo2";
  print("x in foo2: " + x);
  print("y in foo2: " + y);
}

void foo3(){
  print("i in loop: ");
  for(var i = 0; i < 3; i++){
    //i is accessible inside block
    var a = i;
    print( i.toString() + ",");
  }
  //i inaccessible
  //print("i outside loop block in foo3: " + i);
  //a inaccessible outside block
  //print("a outside loop block in foo3: " + a);
}
```

- **OUTPUT:**

```
x in main: main
y in main: global
x in foo2: foo2
y in foo2: global
x in foo1: main->foo1
y in foo1: global
x in main2: main->foo1
x in foo2: foo2
y in foo2: global->foo1
i in loop:
0,
1,
2,
```

- **ANALYSIS:**

Dart is a static scoped language that follows the lexical order for scoping variables. So, although it is possible to access the variable in parent scopes by not defining a type to the variable (ie: referencing a variable instead of declaring it), it is not possible to readily access the variable in dynamic parent of the current block. Also it is not possible to access the global variables if a parent variable with same name hides/ overrides it. So, it is possible to access/ edit the first available variable in parent scopes as shown in the code. Finally, variables declared in control/ loop blocks are not available outside the block. Also, dart supports nested scoping and function declaration.

• Javascript:

- **CODE:**

```
<!DOCTYPE html>
<head>
        <script type="text/javascript">
                //global variable declarations
                var x = "global";
                var y = "global";
                function main() {
                        var x = "main";
                        //should output the variable defined in scope of main()
                        document.write("x in main: " + x + "<br>");
```

```
        //should output the y defined in global scope
        document.write("y in main: " + y + "<br>");

        function foo1() {
                //editing x variable in parent (ie: main)
                x = "main->foo1";

                document.write("x in foo1: " + x + "<br>");
                //prints global y because no y in parent
                document.write("y in foo1: " + y + "<br>");
        }
        foo2();
        foo1();

        //should output the edited x defined in scope of main()
        document.write("x in main2: " + x + "<br>");

        foo2();
}

function foo2() {
        //since static scoping, global y printed
        //local x printed
        var x = "foo2";
        document.write("x in foo2: " + x + "<br>");
        document.write("y in foo2: " + y + "<br>");
}

function foo3() {
        document.write("i in loop: ");
        for (var i = 0; i < 3; i++) {
                //i is accessible inside block
                var a = i;
                document.write(i + ',');
        }
        document.write("<br>")
        //i accessible
        document.write("i outside loop block in foo3: " + i + "<br>");
```

```
                        //a accessible outside block
                        document.write("a outside loop block in foo3: " + a + "<br>");
                }


                main();
                document.write("<br>")
                foo3();
        </script>
</head>
</html>
```

○ OUTPUT:

```
x in main: main
y in main: global
x in foo2: foo2
y in foo2: global
x in foo1: main->foo1
y in foo1: global
x in main2: main->foo1
x in foo2: foo2
y in foo2: global

i in loop: 0,1,2,
i outside loop block in foo3: 3
a outside loop block in foo3: 2
```

○ ANALYSIS:

JavaScript is a static scoped language that follows the lexical order for scoping variables. So, although it is possible to access the variable in parent scopes by not adding a “var” to the variable accessing/ manipulation statement (ie: referencing a variable instead of declaring it), it is not possible to readily access the variable in dynamic parent of the current block. Also it is not possible to access the global variables if a parent variable with same name hides/ overrides it. So, it is possible to access/ edit the first available variable in parent scopes as shown in the code. Finally, variables declared in control/ loop blocks are available outside the block and can be used/ manipulated accordingly. Also, JavaScript supports nested scoping and function declaration.

- **PHP:**
  - **CODE:**

```
<!DOCTYPE html>
  <head>
    <title>Php Scoping</title>
  </head>
  <body>
    <?php
      //global variable declaration
      $x = "global";
      $y = "global";

      function main(){
        //redeclares x as no way to access found in PHP
        $x = "main";

        //should output the variable defined in scope of main()
        echo "x in main: $x\n";
        //should output null as y not found in scope
        echo "y in main: $y\n";

        foo2();
        foo1();

        //should output the original value of x as not edited by children scopes
        echo "x in main2: $x\n";

        foo2();
      }

      function foo1(){
        //redeclares x as no way to access found in PHP
        $x = "foo1";
        #y variable is global y
        global $y;

        //x referenced from foo1, y is from global
```

```
        echo "x in foo1: $x\n";
        echo "y in foo1: $y\n";
        //y declared now, is available now
        $y = "foo1";
      }

      function foo2(){
        //since no nested scoping, y is null and x is redefined
        $x = "foo2";
        echo "x in foo2: $x\n";
        echo "y in foo2: $y\n";

      }

      function foo3(){
        echo "i in loop: ";
        for ($i = 0; $i < 3; $i++){
          //$i is accessible inside block
          $a = $i;
          echo $i . ',';
        }
        echo "\n";
        //$i accessible
        echo("\$i outside loop block in foo3: " . $i . "\n");
        //$a accessible outside block
        echo("\$a outside loop block in foo3: " . $a . "\n");
      }

      main();
      echo "\n";
      foo3();
    ?>
  </body>
</html>
```

○ OUTPUT:

```
x in main: main
y in main:
```

```
x in foo2: foo2
y in foo2:
x in foo1: foo1
y in foo1: global
x in main2: main
x in foo2: foo2
y in foo2:


i in loop: 0,1,2,
$i outside loop block in foo3: 3
$a outside loop block in foo3: 2
```

- **ANALYSIS:**

PHP is a static scoped language that does not allow parent/ nested scopes (with the exception of the ability to access the global variable). So, it is not possible to access the variable in dynamic/ static parent of the current block. But, it is possible to access global variables by using the “global” keyword before a variables name in a block. Once declared as global, all edits to variable alter the global variable as shown in the code output above. Finally, variables declared in control/ loop blocks are available outside the block as well and can be manipulated accordingly. Finally, PHP does not support nested scoping and/or function declaration.

• Perl – static:

- **CODE:**

```
#global variable declaration
$x = "global";
$y = "global";


sub main(){
  my $x = "main";


  #should output the variable defined in scope of main()
  print "x in main: $x\n";
  #should output the y defined in global scope
  print "y in main: $y\n";
```

```
  foo2();
  foo1();

  #should output the edited x defined in scope of main()
  print "x in main2: $x\n";
  foo2();

}

sub foo2(){
  #y from global printed
  my $x = "foo2";
  print "x in foo2: $x\n";
  print "y in foo2: $y\n";

  sub foo1(){
    #editing x variable in static parent (ie: foo2)
    $x = "main->foo1";

    #x referenced from foo2 because static scoping
    #y referenced from global (main's parent scope)
    print "x in foo1: $x\n";
    print "y in foo1: $y\n";
    #edits y in parent scope (ie: global)
    $y = "global->foo1";
  }
}

sub foo3(){
  print "i in loop: ";
  for( $i = 0; $i < 3; $i++){
    #i is accessible inside block
    my $a = $i;
    print "$i,";
  }

  print "\n";
  #i inaccessible
```

```
  print "i outside loop block in foo3: $i\n";
  #a inaccessible outside block
  print "a outside loop block in foo3: $a\n";
}

main();
print "\n";
foo3();
```

- **OUTPUT:**

```
x in main: main
y in main: global
x in foo2: foo2
y in foo2: global
x in foo1: main->foo1
y in foo1: global
x in main2: main
x in foo2: foo2
y in foo2: global->foo1

i in loop: 0,1,2,
i outside loop block in foo3: 3
a outside loop block in foo3:
```

- **Perl – dynamic:**
  - **CODE:**

```
#global variable declaration
$x = "global";
$y = "global";

sub main(){
  local $x = "main";

  #should output the variable defined in scope of main()
  print "x in main: $x\n";
  #should output the y defined in global scope
```

```
  print "y in main: $y\n";

  foo2();
  foo1();

  #should output the edited x defined in scope of main()
  print "x in main2: $x\n";
  foo2();
}

sub foo2(){
  #y from global printed
  local $x = "foo2";
  print "x in foo2: $x\n";
  print "y in foo2: $y\n";

  sub foo1(){
    #editing x variable in dynamic parent(ie: main)
    $x = "main->foo1";

    #x referenced from main because dynamic scoping
    #y referenced from global (main's parent scope)
    print "x in foo1: $x\n";
    print "y in foo1: $y\n";
    #edits y in parent scope (ie: global)
    $y = "global->foo1";
  }
}
sub foo3(){
  print "i in loop: ";
  for( $i = 0; $i < 3; $i++){
    #i is accessible inside block
    local $a = $i;
    print "$i,";
  }

  print "\n";
  #i accessible
```

```
  print "i outside loop block in foo3: $i\n";
  #a inaccessible outside block
  print "a outside loop block in foo3: $a\n";
}
main();
print "\n";
foo3();
```

○ OUTPUT:

```
x in main: main
y in main: global
x in foo2: foo2
y in foo2: global
x in foo1: main->foo1
y in foo1: global
x in main2: main->foo1
x in foo2: foo2
y in foo2: global->foo1

i in loop: 0,1,2,
i outside loop block in foo3: 3
a outside loop block in foo3:
```

○ ANALYSIS:

Perl language allows both static and dynamic scoping by the use of “my” and “local” keywords. So, if a variable is declared as local, apart from being available in the scope, it hides any global/ parent variables and all calls to that variable in children scopes (dynamic children) are directed to this variable. So, “local” keyword enforces dynamic scoping my making the variable accessible to functions called in the current scope.

On the other hand, using “my” keyword before a variable makes that variable accessible in the current scope but also enforces static scoping by redirecting the variable call in the function to its static parent (ie: lexical scoping). Hence, it is possible to have both dynamic and static scoping in perl.

But, once a variable in parent uses either “my” or “local” keyword, it is not possible to alternate the use of variables available in dynamic/ static scopes and only one scope is accessible. Finally, variables declared in control/ loop blocks are not available outside the block. Also, Perl supports nested scoping and function declaration.

CONCLUSION

A quick comparison of five language demonstrates that it is only possible to have dynamic scoping in Perl while all languages utilize static scoping. Apart from Python and PHP, that have the “global” that allow the availability of global variables in any scope, no other language makes the global variables available in children scopes unless they’re not available as parent scope (Apart from PHP).

PHP language does not allow for nested scoping and does not allow accessing variable event from static/ dynamic scopes. Whereas all other languages allow children scopes to access variables from static parents (Python uses nonlocal keyword to do that). Finally, as demonstrated in all codes, if access to parent variables is possible, it not only possible to access the variables but also to manipulate them. Also, With the exception of Perl and Dart, it is possible to access variables declared in loop/ control statement blocks outside the block and manipulate them.