**Dynamic Scope in Perl:**

print "Dynamic Scope for perl \n";

local $x;

sub A{

print "1th decleration for \$x: $x \n";

}

sub B{

$x = 10;

A();

print "2nd decleration for \$x after calling A: $x \n";

}

sub C{

local $x = 5;

B();

print "3th decleration for \$x after calling A: $x \n";

}

$x = 21;

print "Decleration without using 'local' before calling functions: $x \n";

C();

print "Decleration without using 'local' after calling functions: $x \n";

**Output:**

Dynamic Scope for perl

Decleration without using 'local' before calling functions: 21

1th decleration for $x: 10

2nd decleration for $x after calling A: 10 //important

3th decleration for $x after calling A: 10 //important

Decleration without using 'local' after calling functions: 21//important part

**Explanation:** In dynamic scope for Perl language, we need to use ‘local’ term to decleare variables. I tried to show this progress by using three different options. First, I created a local variable without giving any number. Then I created a variable without ‘local’ term. Finally, I created a new local x variable by giving an integer. To show this progress, I used three different sub functions and called them one by one. Before calling sub functions, all of them used its own x variables. However, after calling functions, especially sub C function, they checked whether there is a local term or not. In my output section, there are three important parts which are given by comment. These parts, especially first two, output 10 for x variable because of local situation. Then the last output is 21 because sub functions are done and then x variable didn’t change. Using local term just change functions’ variables.

**Static Scope in Perl:**

#!/usr/bin/env perl

use v5.10.0;

use warnings;

use strict;

say "Static Scope for perl";

my $x;

sub A{

print "1th decleration for \$x: $x \n";

}

sub B{

my $x = 10;

A();

print "2nd decleration for \$x after calling A: $x \n";

}

sub C{

$x = 5;

B();

print "3th decleration for \$x after calling A: $x \n";

}

$x = 21;

print "Decleration without using 'my' before calling functions: $x \n";

C();

print "Decleration without using 'my' after calling functions: $x \n";;

**Outputs:**

Static Scope for perl

Decleration without using 'my' before calling functions: 21

1th decleration for $x: 5

2nd decleration for $x after calling A: 10 //important part

3th decleration for $x after calling A: 5 //important part

Decleration without using 'my' after calling functions: 5//important part

**Explanation:** In static scope for Perl language, we need to use 'my' term to declare variables. I tried to show this progress by using three different options. First, I created a my variable without giving any number. Then I created a new my x variable by giving an integer. Finally, I created a variable without ‘my’ term. To show this progress, I used three different sub functions and called them one by one. Before calling sub functions, all of them used its own x variables. However, after calling functions, especially sub C function, they checked whether there is a local term or not. In my output section, there are three important parts which are given by comment. These parts output different integers rather than dynamic scope. The reason is that if we are using my, we are trying to keep this variable for the rest of the program. That’s why the middle and the last outputs differ from the dynamic scope.

To conclude, I tried to show the differences between local and my usage in Perl language. As we can see that outputs are different even if I used same structure of language.

**Scope in Python:**

print("Python code to show scoping rule");

x = 15;

def f1():

x = 3;

print("First print in f1 %d" % x);

x = 10;

print("Second print in f1 %d" % x);

def f2():

global x;

print("First print in f2 %d" % x);

x = 1;

print("Second print in f2 %d" % x);

def f3():

x = 0;

print("First print in f3 %d" % x);

def f4():

global x;

print("First print in f4 %d" % x);

x = 315;

print("First print in f4 %d" % x);

print("Before calling f4 %d" % x);

f4();

print("After calling f4 %d" % x);

print("Before calling f3 %d" % x);

f3();

print("After calling f3 %d" % x);

print("Before calling f2 %d" % x);

f2();

print("After calling f2 %d" % x);

print("Before everything %d" % x);

f1();

print("After everything %d" % x);

**Outputs:**

Before everything 15

First print in f1 3

Second print in f1 10

Before calling f2 10

First print in f2 15

Second print in f2 1

Before calling f3 1

First print in f3 0

Before calling f4 0

First print in f4 1

First print in f4 315

After calling f4 0

After calling f3 315

After calling f2 10

After everything 315

**Explanation:** In Python language, we are using global term to show scoping rule. First of all I created four different functions and called them one by one to show how a variable can change if we used global term at the beginning of declaration part. I used more than one global variable to show changing operation. First and the last variables are declared as normal integer. However, in function 4 and 2, I used global variables after print and call some normal functions which contain normal variables. After all these progress I saw that if we are using global variables in Python, it affects the following functions only. That’s why the last part which is called “After everything” is assigned like the last global variable (315).

**Java Script Code for Scope**

<html>

<body>

<script>

var x = document.write("First decleration for x before functions<br>");

function A(){

var y = document.write("First decleration for y in function A<br>" );

function B(){

var z = document.write("First decleration for z in function B<br>");

x = document.write("Second decleration for x in function B<br>");

y = document.write("Second decleration for y in function B<br>");

return 1;

}

B();

return 1;

}

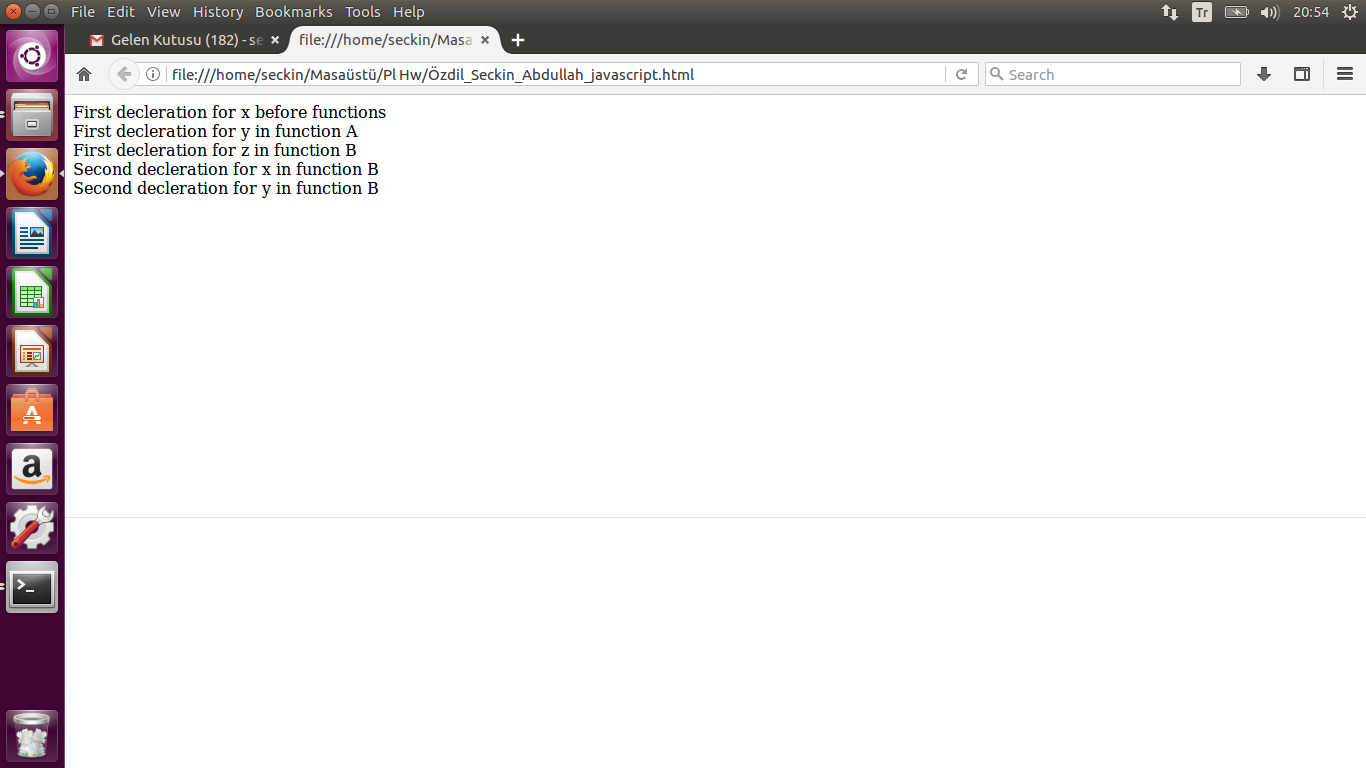
A();

</script>

</body>

</html>

**Output:**



**Explanation:** In Java Script, we need to use ‘var’ term to show scope. In my example, I created two functions and their variables. After all of calling operations only three outputs appeared. The reason is that ‘var’ can see the last variable in the code segment. I put the output screen in report for Java Script.

**PHP Code for Scope**

<?php

echo "Scope by Using PHP";

$x = 45;

$y = 50;

function A(){

echo "First declaration for x is $GLOBALS[$x] \n";

echo "First declaration for y is $GLOBALS[$y] \n";

$x = 10;

$y = 5;

echo "Second declaration for x is $x \n";

echo "Second declaration for y is $y \n";

}

A();

?>

**Output:**

First declaration for x is

First declaration for y is

Second declaration for x is 10

Second declaration for y is 5

**Explanation:** We need to use $GLOBALS term to show scope process in PHP. First, I created two different variables and then call the function by using $GLOBALS. As we can see that no output appears. However, after second declaration part 10 and 5 appear immediately.