**Python:** (Please run the python file using python3)

**File name:** haque\_faaiz\_ul\_python.py

**Code:**

a = 5 #a is global

def f1():

a = 3 #a is local to f1

b = 1 #b is local to f1

c = 0 #c is local to f1

print ("a:", a, " b:", b, " in f1" ) #a: 5 b: 1

def f2():

global a #a is assigned as global

a = 3 #a now changes global a

b = 3 #still local variable in f1

print ("a:", a, " b:", b, " in f2 " ) #a: 3 b: b

def f3():

nonlocal c #nonlocal so can access most outer c

c = 3 #c in f1 changes from 0 to 3

print ("c:", c, " in f3 " ) #c: 3

f3();

f2();

print("c:", c, " in f1 " ) #c: 3

print ("a:", a, " in main ")

f1();

print ("a:", a, " in main after calling f1 ")

**Outputs of the program:**

*a: 5 in main*

*a: 3 b: 1 in f1*

*a: 3 b: 3 in f2*

*c: 3 in f3*

*c: 3 in f1*

*a: 3 in main after calling f1*

**Explanations:**

In f1(): Variable b is local to f1, this is an example of static scoping. We can only access b in its current static scope which has a reference environment in f1. When we print a and b we get the outputs as 5 and 1. 5 from the global value of a, and 1 from the local value of b.

In f2(): We declare a as global here. Now changing the value of a will be in reference to the a declared outside the functions, the global a. So if we were to print the global a after all function calls we will obtain the value of 3 in this case. Since f2’s definition is in the scope of f1 we can access the local variable to f1 in f2 and print it with its value or even change it. So it outputs to 3.

Two important things to note here:

1) LIMITATION: If we were to try and print a before declaring a as global it would give the following error message:

SyntaxError: name 'a' is used prior to global declaration

This is because we can not both access the old global a and declare a as a new global variable at the same time. An example of this code would be:

a = 5 #a is global

{..}

def f2():

print (a)

global a

a = 3

{…}

2) ADVANTAGE: We can still define a variable a without using the global keyword in f2(), but in this case it would simply just locally be defined to f2.

The global keyword is a huge advantage and allows convenient programs in Python.

In f3(): Here we define a nonlocal variable c. Changing this variable will change the most recent parent where local c was defined. So in this case c is found in f1 only, so the old value in f1 of c will change to 3 in this case. Now c is 3 in both f3 and f1. This is very advantageous and can be beneficial in some programs. A limitation in this case if c were to be defined globally, we would not be able to change the value. So nonlocal variables can only change values of local variables. Example:

global a = 5

def f1():

nonlocal a

a = 3 // Error msg: no binding for nonlocal ‘a’ found

In main: we cannot access the variables b and c since they were locally defined. WE can only access the global variable a. Attempting to access variables that are not global in this scope will return an error. Before calling f1() a is 5, and after calling f1() we call f2() and in f2 we assign global a and change its value to 3. So a is initially 5 then changes to 3 after function calls.

**Perl (static)**

**File Name:** haque\_faaiz\_ul\_perl\_static.pl

**Code:**

$a = 5; #a is global

sub f1 {

my $a = 2; #static scoping, a is local to f1

print "a=$a in f1\n";

f2();

}

sub f2 {

print "a=$a in f2\n"; #reference environment is only global

$a = 7; #accesses global variable of a

f3();

}

sub f3 {

my $b = 0; #local variable is statically scoped to f3

print "b=$b in f3\n";

$b = 3;

}

print "a=$a b=$b in main \n"; #a = 5 from global and b not applicable

f1();

print "a=$a b=$b in main after calling functions\n";

#a = 7 since we changed the global value of a b still not applicable as local **Outputs:**

*a=5 b= in main*

*a=2 in f1*

*a=5 in f2*

*b=0 in f3*

*a=7 b= in main after calling functions* **Explanations:**

In f1(): we create a local variable using the my keyword. This is static scoping. We then print and simply get the value assigned.

In f2(): We do not have any local value a assigned so when we attempt to print $a it gets the global value $a and prints it. After this we set $a = 7 and it assumes $a is referring to the global since no keyword is specified. This is advantageous for the program.

In f3(): Another local variable $b is created and simply printed. After printing, we attempt to change b’s value again but since $b was defined as my it is still local to f3 and does not become a global variable.

Now in main before the function calls b is not defined yet and $a takes its global value. And after the function calls a’s value is changed in $f2 but b is still not available due to it never being defined as a global function. There are few limitations here in the static scoping of perl programs.

**Perl (dyanmic)**

**File Name:** haque\_faaiz\_ul\_perl\_dynamic.pl

**Code:**

$a = 10; #global

sub f1 {

my $a = 2; #static

local $b = 3; #dynamic

print "a=$a b=$b in f1 \n"; #a = 2 b = 2

f2();

}

sub f2 {

print "a=$a b=$b in f2 \n"; #a = 10 b = 3

f3();

}

sub f3 {

$b = 7; #still local

print "a=$a b=$b in f3 \n"; #a = 10 b = 3

}

f1();

print "a=$a b=$b in main \n"; #a = 10 b =

**Outputs:**

*a=2 b=3 in f1*

*a=10 b=3 in f2*

*a=10 b=7 in f3*

*a=10 b= in main*

**Explanations:**

In f1(): $a is static and local to f1 while $b is dynamic. We can print from the two values set in f1 and obtain 2 and 3 respectively

In f2(): Since f1() called f2() the dynamic variable created in f1 is also obtainable in f2(), but the static variable isn’t. So when we print $a and $b we obtain 10 and 3, where $a is taken its global value and $b is taken its value in f1().

In f3(): We attempt to change value of b and are able to but this changes the value in f1() since f3 is called by f2 and f2 is called by f1, in this case b is still dynamic and local. However, a was static in f1 and therefore its global value is retained.

In main: We can not access the value of b since it was declared in the scope of f1 only. Value of $a is still taken from global

PHP:

File Name: haque\_faaiz\_ul\_php.php

**Code:**

<?php

$a = 5; //Global

$b = 3; //Global

function f1() {

$a = 3; //Local

print "a=$a in f1\n"; #a = 3

f2();

}

function f2() {

$c = $GLOBALS['b']; #accesses variables from global scope

print "c=$c in f2\n"; #c= 3

f3();

}

function f3() {

global $a; #a defined as global

$a = 2; #changes old global value

print "a=$a in f3 \n"; #a = 2

}

f1();

print "a=$a b=$b in main\n"; #a=2 b=3

?>

**Outputs:**

*a=3 in f1*

*c=3 in f2*

*a=2 in f3*

*a=2 b=3 in main*

**Explanations:**

In f1(): $a is defined as a local variable to f1 and simply prints a in f1 as 3.

In f2(): $c accesses a variable $GLOBALS which has access to all variables defined in local scope. $c takes the value of $b in this case and we simply output value of 3. This is very advantageous to easily access global variables.

In f3(): We define $a as global and are able to change the value of the old global variable of $a. We set it to 2 and print 2

In main: Before execution of functions global values of a and b are 5 and 3 respectively. But since we call f1 which calls f2 which calls f3 and we change global value of a to 2 , we get 2 and 3 after function calls.

A limitation in php if we simply try to print $a in f1 or f2 or f3, it does not recognize the global so we are forced with the restriction of defining a as global first in order to print it.

**Javascript:**

**File Name:** haque\_faaiz\_ul\_javascript.html

**Code:**

<html>

<head>

JavaScript Examples <br>

<script>

function f1() {

var a; //local to f1

a = 4;

c = 10; //global

function f2() {

var b; //local to f2

b = 3;

alert("a= " + a + " b= "+ b + “ in f2 “); //a= 4 b = 3

}

f2();

}

function f3() {

alert("c= " + c + “in f3” );

}

</script>

</head>

<body>

<button type="button" onclick="f1()">call f1()</button>

<button type="button" onclick="f3()">call f3()</button>

</body>

</html>

**Outputs:**

*After calling f1(): a= 4 b= 3 in f2*

*After calling f3(): c= 10 in f3*

**Explanations:**

In f1(): var a is local to f1 and statically scoped. C is not specified with the var keyword and is assumed to be global. F1 calls f2

In f2(): Since we are still in scope of f1 we can access the local variable a, and print it. Var b is local to f2.

In f3(): We can access the variable c even though we are no longer in scope of f1 where it was declared, this is because it is presumed to be global without using a keyword. Ambiguous to new users but advantageous for experts to use this global concept.