Department of Computer Engineering

University of Peradeniya

CO523 - Programming Languages

Lab 04: Demonstrating Static vs Dynamic Scoping

E/20/449
Wijewardhana A.R.S.S.

Task 1

Run the following python code, and explain the output.

```
a = 5

def f():
  print(a)

def g():
  a = 10
  f()

g()
```

Output :5

- **Why is it 5 and not 10?**
    - Python uses **Static Scoping** (also called Lexical Scoping).
    - In static scoping, a variable's value is determined by the **program text structure** (where the function is *defined*), not the runtime call stack (where the function is *called*).
    - The function f() is defined in the global scope where a = 5.
    - Even though f() is called inside g() (where a is locally set to 10), f() cannot see g()'s local variables. It looks for a in its own definition environment, finds the global a, and prints **5**.

Task 2

Convert the Python code from **Task 1** into Perl syntax. You will perform a two-part experiment by modifying the variable declaration inside the function g().

Case A: Using `'my'` Keyword

Case B: Using `'local'` Keyword

Run the code and record the output. Explain the output for two cases.

Case A: Using 'my' Keyword (Static Scoping)

```
1  $a = 5;
2
3  sub f {
4      print "$a\n";
5  }
6
7  sub g {
8      my $a = 10;  # Using 'my'
9      f();
10 }
11
12 g();
```

STDIN

Input for the

Output:

5

- Why 5? You used the my keyword. This enforces Static (Lexical) Scoping.
- When f() runs, it needs to find $a.
- Because of static scoping, f() looks at where it was defined (the global scope), not where it was called.
- It ignores the $a = 10 inside g() because my keeps that variable private to g()'s physical block .
- Therefore, f() prints the global value: 5.

Case B: Using 'local' Keyword (Dynamic Scoping)

```
1  $a = 5;
2
3  sub f {
4      print "$a\n";
5  }
6
7  sub g {
8      local $a = 10;  # Using 'local'
9      f();
10 }
11
12 g();
```

STDIN

Input for the prog

Output:

10

- Why 10? You used the local keyword. This enforces Dynamic Scoping.
- When f() runs, it looks for $a.
- Under dynamic scoping, f() looks at the Call Stack (who called me?).

- Since g() called f(), and g() used local $a = 10, the function f() "sees" this temporary value.
- Therefore, f() prints the value currently active in the call chain: 10.

Task 3

Complete the Perl code below by filling in the blanks with either 'my' (Lexical) or 'local' (Dynamic). For each case, run the code in Perl, and explain the output.

```
$x = "Global";

sub level_3 {
    print "Level 3 sees x as: $x\n";
}

sub level_2 {
    # TO DO
    ______ $x = "Level 2 Value";
    level_3();
}
```

```
sub level_1 {
    # TO DO
    ______ $x = "Level 1 Value";
    level_2();
}

level_1();
```

Cases you have to explain:

Case	Level 1 Key word	Level 2 Keyword
A	my	my
B	local	my
C	my	local
D	local	local

Case A: Level 1 = my, Level 2 = my

```perl
$x = "Global";

sub level_3 {
    print "Level 3 sees x as: $x\n";
}

sub level_2 {
    my $x = "Level 2 Value";  # my
    level_3();
}

sub level_1 {
    my $x = "Level 1 Value";  # my
    level_2();
}

level_1();
```

STDIN

Input for the program (Optior

Output:

```
Level 3 sees x as: Global
```

- **Static Scoping:** Both level_1 and level_2 use my. This creates purely private variables visible **only** inside their own curly braces {}.
- These private variables are **not** visible to level_3.
- When level_3 runs, it looks for $x. Since no one dynamically replaced the global variable, level_3 finds the original $x = "Global".

Case B: Level 1 = local, Level 2 = my

```perl
$x = "Global";

sub level_3 {
    print "Level 3 sees x as: $x\n";
}

sub level_2 {
    my $x = "Level 2 Value";  # my
    level_3();
}

sub level_1 {
    local $x = "Level 1 Value";  # local
    level_2();
}

level_1();
```

STDIN

Input for the program (Optional)

Output:

```
Level 3 sees x as: Level 1 Value
```

- level_1 uses local. This temporarily replaces the Global $x with "Level 1 Value".
- level_1 calls level_2.
- level_2 uses my. This creates a new, private variable $x = "Level 2 Value". This private variable only exists inside level_2 and does *not* touch the global variable.
- level_2 calls level_3.
- level_3 executes. It cannot see level_2's private my variable. It looks at the Global $x.

- The Global $x is currently holding "Level 1 Value" (because level_1 changed it dynamically).

Case C: Level 1 = my, Level 2 = local

```perl
$x = "Global";

sub level_3 {
    print "Level 3 sees x as: $x\n";
}

sub level_2 {
    local $x = "Level 2 Value";  # local
    level_3();
}

sub level_1 {
    my $x = "Level 1 Value";  # my
    level_2();
}

level_1();
```

STDIN

Input for the program (Optional)

Output:

```
Level 3 sees x as: Level 2 Value
```

- level_1 uses my. It creates a private variable "Level 1 Value". The Global $x remains "Global".
- level_1 calls level_2.
- level_2 uses local. It temporarily replaces the Global $x with "Level 2 Value".
- level_2 calls level_3.
- level_3 executes. It looks at the Global $x.
- The Global $x is currently holding "Level 2 Value" (because level_2 changed it dynamically).

Case D: Level 1 = local, Level 2 = local

```perl
$x = "Global";

sub level_3 {
    print "Level 3 sees x as: $x\n";
}

sub level_2 {
    local $x = "Level 2 Value";  # local
    level_3();
}

sub level_1 {
    local $x = "Level 1 Value";  # local
    level_2();
}

level_1();
```

STDIN

Input for the program (Optional)

Output:

```
Level 3 sees x as: Level 2 Value
```

- level_1 uses local. Global $x becomes "Level 1 Value".

- level_1 calls level_2.
- level_2 uses local. It saves the previous value ("Level 1 Value") and temporarily sets Global $x to "Level 2 Value".
- level_2 calls level_3.
- level_3 executes. It looks at the Global $x.
- The Global $x is currently holding "Level 2 Value".

Summary

Case	Level 1	Level 2	Output	Reason
A	my	my	Global	Both are private; Global variable is untouched.
B	local	my	Level 1 Value	L1 updates Global; L2 creates a private var that L3 ignores.
C	my	local	Level 2 Value	L1 is private; L2 updates Global, which L3 sees.
D	local	local	Level 2 Value	L1 updates Global; L2 updates Global again (most recent change wins).