Lab 04: Demonstrating Static vs Dynamic Scoping – E/20/037

Task 1

Run the following python code, and explain the output.

```
a = 5

def f():
  print(a)

def g():
  a = 10
  f()

g()
```

- Python uses **static (lexical) scoping**.
- When f() executes, it looks for variable a in its lexical environment (where it was defined in the source code), not where it was called from.
- Since f() was defined at the global level, it refers to the global a = 5.
- The local a = 10 inside g() is not visible to f() because f() was not defined inside g().
- In static scoping, variable binding is determined by the physical structure of the code, not the call stack.

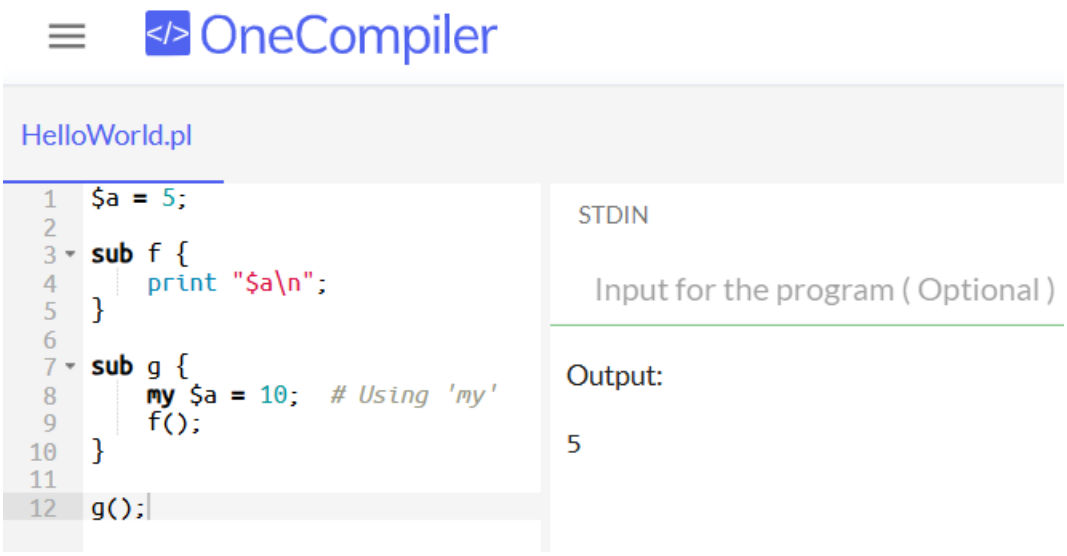
Task 2

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.



The ‘my’ keyword creates a **lexically scoped** variable.

This means my $a = 10 creates a NEW variable $a that only exists within the g() subroutine's lexical scope.

When f() is called, it looks for $a in its lexical environment (where it was defined), which is the global scope.

So it prints the global $a = 5. This demonstrates **static scoping**.

The local keyword creates a **dynamically scoped** variable.

It temporarily modifies the global $a for the duration of g()'s execution and any subroutines called from within g().

When f() is called from g(), it sees the temporarily modified value $a = 10 because dynamic scoping looks at the **call stack** at runtime, not the lexical structure.

This demonstrates **dynamic scoping**.

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();
}
```

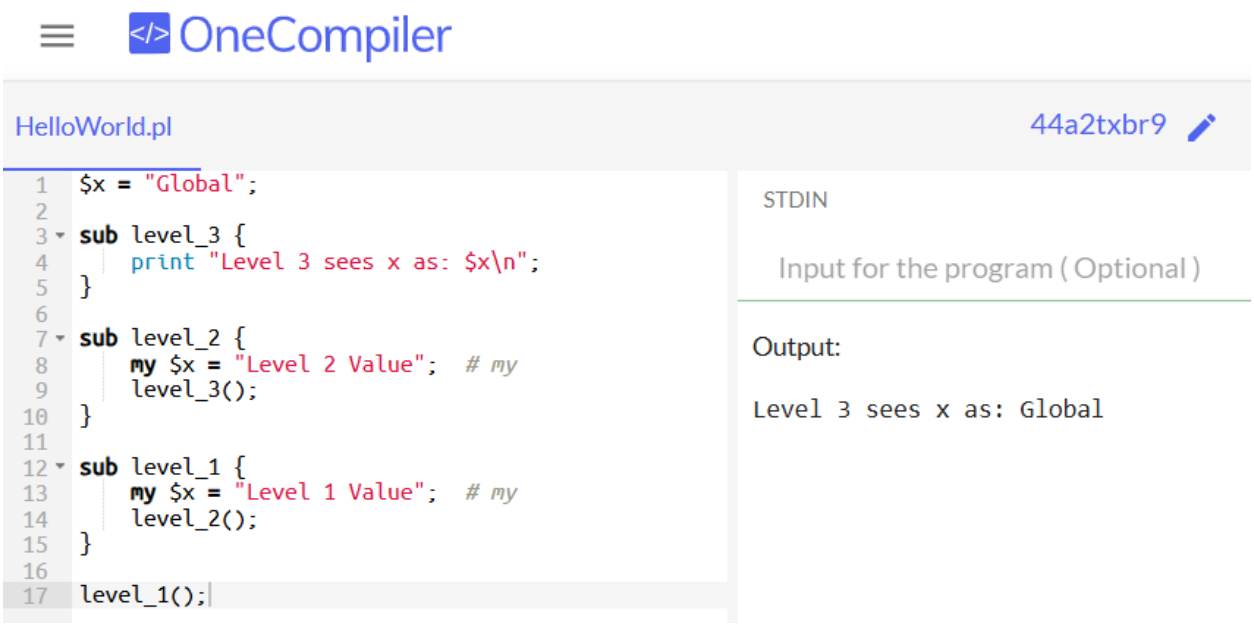
```
sub level_1 {
    # TO DO
    ______ $x = "Level 1 Value";
    level_2();
}

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



Both my keywords create **lexically scoped variables**. Each creates a new variable $x that only exists within its respective subroutine. When level_3() executes, it looks for $x in its lexical scope (where it was defined), which is the global scope. It cannot see the lexically scoped variables in level_1 or level_2 because they are isolated to their own scopes.

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

level_1 uses local, which **temporarily modifies the global $x** to "Level 1 Value" for the **duration of the call chain**. However, level_2 uses my, which creates a NEW lexically scoped variable that only exists within level_2 itself. When level_3() is called, it looks at the **global $x which has been temporarily modified by the local in level_1**.

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

level_1 uses my, creating a lexically scoped variable that doesn't affect the global $x. **level_2** uses local, which **temporarily modifies the global $x** to "Level 2 Value". When level_3() executes, it sees this local modification in the call stack. Result: prints "Level 2 Value".

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

Both use local, creating dynamically scoped temporary modifications to the global $x. level_1 first sets it to "Level 1 Value", then level_2 sets it to "Level 2 Value". Since **local creates temporary values that are visible in the call stack**, and **level_2's modification is more recent** (deeper in the call stack), level_3() sees "Level 2 Value".

'my' creates isolated lexical scopes (static scoping), while 'local' temporarily modifies the global variable for the duration of the call chain (dynamic scoping).