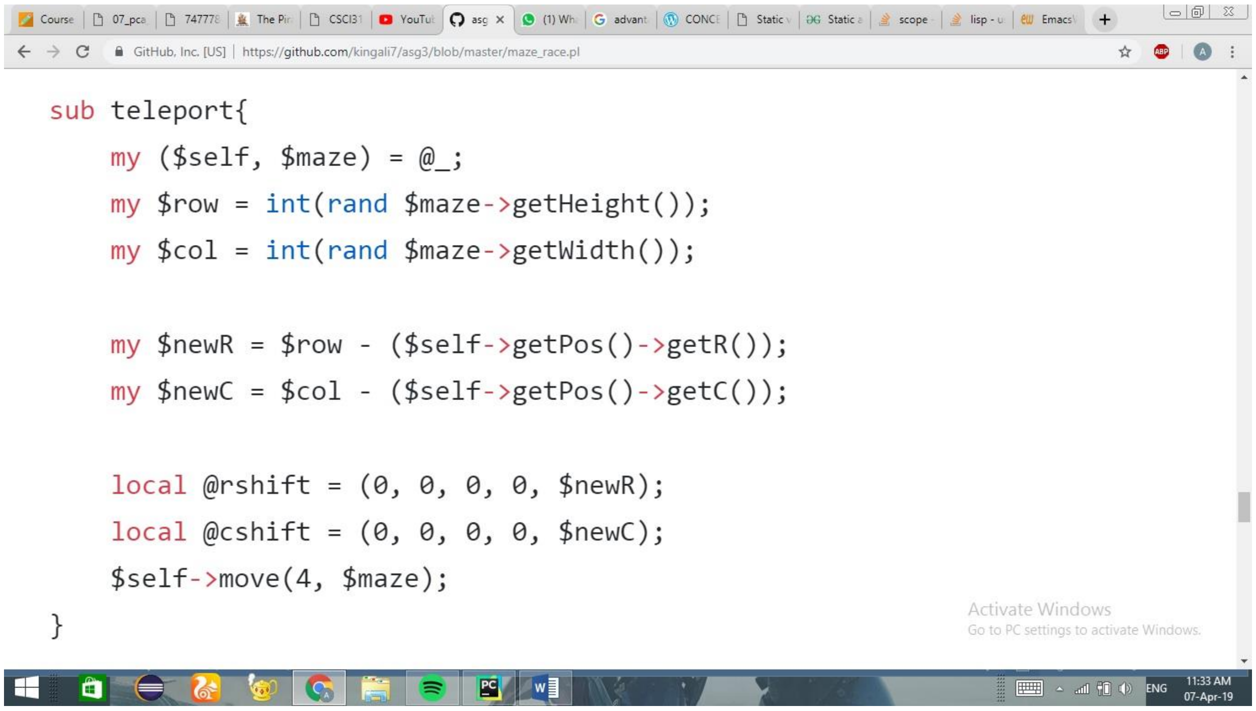
Q1. Provide example code and necessary elaborations for demonstrating the advantages of Dynamic Scoping in using Perl to implement Maze Race as compared to the corresponding codes in Python.

Perl Code:



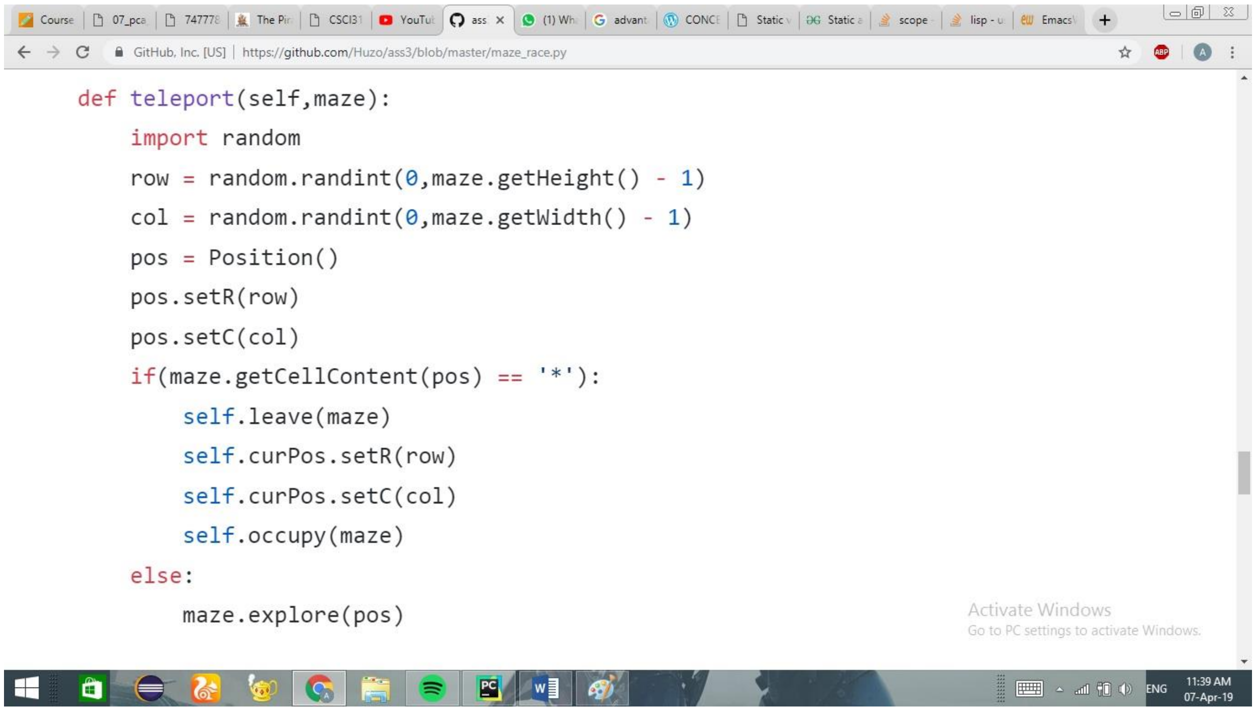
GitHub, Inc. [US] | https://github.com/kingali7/asg3/blob/master/maze_race.pl

```perl
sub teleport{
    my ($self, $maze) = @_;
    my $row = int(rand $maze->getHeight());
    my $col = int(rand $maze->getWidth());

    my $newR = $row - ($self->getPos()->getR());
    my $newC = $col - ($self->getPos()->getC());

    local @rshift = (0, 0, 0, 0, $newR);
    local @cshift = (0, 0, 0, 0, $newC);
    $self->move(4, $maze);
}
```

Python Code:



GitHub, Inc. [US] | https://github.com/Huzo/ass3/blob/master/maze_race.py

```python
def teleport(self,maze):
    import random
    row = random.randint(0,maze.getHeight() - 1)
    col = random.randint(0,maze.getWidth() - 1)
    pos = Position()
    pos.setR(row)
    pos.setC(col)
    if(maze.getCellContent(pos) == '*'):
        self.leave(maze)
        self.curPos.setR(row)
        self.curPos.setC(col)
        self.occupy(maze)
    else:
        maze.explore(pos)
```

Like everything else, Dynamic Scoping is merely a tool. Used well it can make certain tasks easier. When used poorly it can introduce bugs and headaches. I can certainly see some uses for it. For example it can eliminate the need to pass variables to some functions. Let's compare codes, fifth element in the rshift and cshift are for to make moves like rush, block-through and teleport. After calling move function in perl code, move function uses updated rshift and cshift arrays in teleport function to make moves, because those array variables are dynamically scoped. Therefore, there is no need pass updated variables into move function or to do it manually like the one in the Python code.

Q2. Discuss the keyword local in Perl (e.g. its origin, its role in Perl, and real practical applications of it) and giving your own opinions.

A **local** gives a temporary value(s) to the global variables. It should be noted that it does not create a local variable and this technique is called dynamic scoping. **Local** is mostly used when the value of a variable has to be visible to the called subroutines. Despite the existence of **my** (which is used more often) there are still several situations where **local** should be used.

1. You need to give a global variable a temporary value. In particular it is important to localize **$_** in all the subroutine that assigns to it.
2. Quite similar to the first point, you should use local when you want to temporarily change only one element of an array.

In my opinion, useful uses of my **local** shine when you would like to use **my**, but you because of some restrictions you can't. For most of the cases, I believe, **my** should be used. Even the most useful cases of local are not very useful.