Q1)

The function “teleport” (Perl on the left, Python on the right)

```perl
sub teleport{
    my ($self, $maze) = @_;
    my $row_rand = int(rand $maze->getHeight());
    my $col_rand = int(rand $maze->getWidth());
    my $row_cur = $self->getPos()->getR();
    my $col_cur = $self->getPos()->getC();

    my $row_dif = $row_rand - $row_cur;
    my $col_dif = $col_rand - $col_cur;

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

```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.

As can be seen the code on the left provides convenience in the way that we can always make use of the move function by merely modifying the dynamically scoped array variables cshift and rshift. On the other hand, this decreases the readability of the code. As can be seen when comparing the Python code to Perl code. So, there is always a tradeoff between convenience and readability.

Q2)

Most important role of **local** keyword is that it temporarily changes the value of global variables which is it is the indication of dynamic scoping once done. In other words, when we try to access the value of a variable, unlike statically scoped variables, we do not go out one level but check the variable’s value in the place it was previously called. Most of the time the keyword local is used if we want the variable to be visible to called subroutines. Although “my” has a broader practice, occasionally using “local” is a cleverer option.

There are some cases the local is useful. Examples are:

1. When you have to assign a temporary value to the global variable. In particular it is important to localize **$_** in all the subroutine that assigns to it.
2. Similar to the first case which is when you have to temporarily change only one element of an array, the one I used in functions like teleport, block-through and rush.

In my humble opinion, local should not be used if possible. Although it can make conveniences in some scenarios as stated above, usually it could cause a lot of un-clarity and confusion to the programmer and the people who read it. Therefore, I think this might be the reason behind why most of the modern programming languages do not use dynamic scoping.