1)

```python
class Jail:
    def __init__(self):
        pass

    def print(self):
        print("Jail ", end='')

    def stepOn(self):

        print ("Pay $1000 to reduce the prison round
        while(True):
            x = input()
            if (x=='y'):
                if(cur_player.money >=1000):
                    cur_player.prison_rounds = 1
                    cur_player.putToJail();
                    break;
                else:
                    print ("You do not have enough m
                    cur_player.prison_rounds = 2
                    cur_player.putToJail();
                    break;
            elif (x == 'n'):
                cur_player.prison_rounds = 2
                cur_player.putToJail();
                break;
            else:
                print ("Please enter [y] or [n].")
```

```python
def main():
    global cur_player
    global num_dices
    global cur_round
    global cur_player_idx

    while terminationCheck():
        cur_round += 1
        cur_player = players[cur_player_idx]

        if (cur_player.num_rounds_in_jail<=0):
            printGameBoard()
            for player in players:
                player.printAsset()

        Player.income = 0
        Player.tax_rate = 0
        Player.due = 200
        Player.handling_fee_rate = 0
        cur_player.payDue()
```

Python

```perl
package Jail;
sub new {
    my $class = shift;
    my $self  = {};
    bless $self, $class;
    return $self;
}

sub print {
    print("Jail ");
}

sub stepOn {

    print "Pay \$1000 to reduce the prison round to
    while(1){
        my $input = <STDIN>;
        chomp $input;
        if ($input eq 'y'){
            if($main::cur_player->{"money"} >= 1000)
                local $Player::prison_rounds = 1;
                $main::cur_player->putToJail();
                last;
            }
            else{
                print "You do not have enough money
                $main::cur_player->putToJail();
                last;
            }
        }
        elsif ($input eq 'n'){
            $main::cur_player->putToJail();
            last;
        }
        else{
            print "Please enter [y] or [n].\n"
```

```perl
sub main {
    my $dice;
    while (terminationCheck()){
        $cur_round = $cur_round + 1;
        $cur_player = $players[$cur_player_idx];

        if($cur_player->{"num_rounds_in_jail"}<=0){
            printGameBoard();
            foreach my $player (@players) {
                $player->printAsset();
            }
        }

        local $Player::tax_rate = 0;
        $cur_player->payDue();
```

Perl

Perl allows dynamic scoping by the keyword “local” while python only allows static scoping. Thus, the identifier in perl will looks up for the most recently valued while the keyword “local” temporarily masking out the previous value but not updating it, therefore I don’t need to reset all the variables that are going to be used by function “payDue()” and only setting the one that is different from the original value.
However, in python, I need to update all of them to ensure the value be called in the function is correct as those value would be updated after resetting it.

Moreover, with dynamic scoping in perl, we can perform function call without passing extra parameters which is also illustrated when calling the method of “payDue”.

These allows us writing shorter code in sub (no need to reset all the parameters) and it is more convenient and generic in changing setting of the code. For example, if we want the prison round increased to 3 for all cases unless player pays $1000 to reduce round to 1, we only need to change the original value is enough. But for python, we need to change all places that have function call involving this parameters.
Moreover, the advantage of dynamic scoping is also the function parameters passing can be ignored which enhance the flexibility in function parameters.

2)

Origin of "local" keyword:

Perl's early version only had global variables. "local" was started be introduced in Perl 4 for localization supporting dynamic scoping. In perl 5, "my" supporting lexical scoping is introduced.

Role of "local" in Perl:

It allows dynamic scoping of a variable for temporarily masking out the previous value but no updating it and also provide the flexibility in parameter passing.

Real practical application of it:

It can make some tasks to be simpler by eliminating the need of resetting all variables and the need of parameter passing, example is this programming assignment 3 illustrated in detail in the previous question.

My opinion:

Though it may be useful for modules having high coupling (though it is not a good performance), it can significantly reduce the effort in resetting the variables back to its default value or new value. But we can just perform resetting and parameters passing in order to provide the same functionality. Therefore, the advantage it provided doesn't seem to be really significant.