```
/*
*CSCI3180 Principles of Programming Languages
*
*--- Declaration ---
*
*I declare that the assignment here submitted is original except for source
*material explicitly acknowledged. I also acknowledge that I am aware of
*University policy and regulations on honesty in academic work, and of the
*disciplinary guidelines and procedures applicable to breaches of such policy
*and regulations, as contained in the website
*http:/ / www . cuhk . edu . hk / policy / academichonesty /
*
*Assignment 3
*Name : YU Si Hong
*Student ID : 1155141630
*Email Addr : shyu0@cse.cuhk.edu.hk
*/
```

1.Example

I used dynamic scoping in `$flag_defeat` and `$flag_rest` to update fighters' properties and award coins to fighters in perl version.

In package AdvancedTournament, it has the following.

```perl
sub update_fighter_properties_and_award_coins {
    my $self    = shift;
    my $fighter = shift;
    our $flag_defeat = shift;
    local $flag_defeat = $flag_defeat;
    our $flag_rest = shift;
    local $flag_rest = $flag_rest;
    $fighter->update_properties();
    $fighter->obtain_coins();
}
```

In package AdvancedFighter, it has the following.

```perl
sub update_properties {
    ...

    if ($AdvancedTournament::flag_defeat) {
        $temp_att = $temp_att + 1;
    }
    if ($AdvancedTournament::flag_rest) {
        $temp_att = 1;
        $temp_def = 1;
        $temp_spe = 1;
    }

    ...
}
```

In python version, when it has no dynamic scoping, I deliver these information by `record_fight` and update fighters' properties accordingly.

```python
def record_fight(self, fight_result):
    if fight_result == "defeat":
        self.history_record.append(fight_result)
        return
    if len(self.history_record) > 0 and self.history_record[-1] == "defeat":
        del self.history_record[-1]
    if len(self.history_record) > 0 and self.history_record[-1] == "rest":
        del self.history_record[-1]
    ...
```

Comparing two implementations, dynamic scoping makes it possible for a method to use a variable from where it was called directly, other than delivering the variable by attaching it with the method's name, like `update_fighter_properties_and_award_coins(fighter, flag_defeat, flag_rest)`.

Once a dynamic scoping variable is defined, it can be used in any methods it will call after that. A advantage is to let progamme be more flexible.

You do not need to taking words to attach the variable with the method's name everytime. A advantage is make the method look more clean.

If you need to deliver a variable through multiple methods, taking words to attach the variable takes effort. A advantage is to reduce workload in some case.

2.Opinion

By using `strict`, I find that to use `local` variable, I need to initialize it as an `our` variable first. The origin of a `local` variable should be `our`. And `our` is visible in a static order while `local` is only visible a dynamic order. And `local` temporarily masks the `our` for all the methods it calls after.

```perl
package cook_a_fish;
our $state = "have fish today\n";
sub task {
    local $state = "fish\n";
    print $state;
    buy();
    print $state;
}
sub buy {
    $state = "fish bought\n";
    wash();
}
sub wash {
    $state = "fish washed\n";
    cook();
}
sub cook {
    $state = "fish cooked\n";
}
task();
print $state;
```

In my opinion, dynamic scoping has its property and it would show its advantages in some cases, like the above perl example to keep delivering the same variable. And I think a flexible language increases the cost of learning it, but that's the fun of programming.