Tutorial 8: Dynamic Scoping Review & PA3

Huancheng Puyang & Yanjing Ren

hcpuyang22@cse.cuhk.edu.hk

yjren22@cse.cuhk.edu.hk

Outline

- ➢Basic Perl Knowledge Review
  - ➢Static vs. Dynamic Scoping

- ➢Programming Assignment 3 Review
  - ➢Task 1: Basic Horse Racing Game
  - ➢Task 2: Advanced Horse Racing Game
  - ➢Implementation Hints
  - ➢Report & Submission

Outline

- Basic Perl Knowledge Review
  - Static vs. Dynamic Scoping

- Programming Assignment 3 Review
  - Task 1: Basic Horse Racing Game
  - Task 2: Advanced Horse Racing Game
  - Implementation Hints
  - Report & Submission

Static and Dynamic Scoping in Perl

- Lexical variable – static scoping
  - Declared with keyword *my*
- Package variables – both static and dynamic scoping
  - Statically scoped package variables
    - Declared with keyword *our*
  - Dynamically scoped package variables
    - Declared with keyword *local*

Package Variables: Comparison

- Rules for static scoping
  - Search in the local function (the function which is running now)
  - Search in the function (or scope) in which that function was defined
  - Search in the function (or scope) in which that function was defined…
  - So forth

  Static Scoping

- Rules for dynamic scoping
  - Search in the local function
  - Search in the function that called the local function
  - Search in the function that called that function…
  - and so on, up the call stack.

  Dynamic Scoping

Outline

- Basic Perl Knowledge Review
    - Static vs. Dynamic Scoping
- Programming Assignment 3 Review
    - Task 1: Basic Horse Racing Game
    - Task 2: Advanced Horse Racing Game
    - Implementation Hints
    - Report & Submission

Task 1: Basic Horse Racing Game

Problem

- Horse Racing Problem
  - User initialize Horse, Team and Court instances;
  - In each round, user provide the racing orders of undefeated horses for both team;
  - Perform the races, examine horse status and update horse attributes;
  - Repeat until one of the teams have all its horses defeated.
- Key Concepts
  - Horse A **defeat** Horse B: the *morale* value of Horse B becomes non-positive;
  - Horse A **wins** the race against Horse B: Horse A has a larger *actual_speed* value.
- Please refer to Part 1 of Assignment Specification for more details.

Task 1: Basic Horse Racing Game

Three Modules

- Horse.pm

  Define the horse instance.

- Team.pm

  Define the team instance.

- Court.pm

  Define the game engine and simulate the racing game.

Task 1: Basic Horse Racing Game

Three Modules: Horse

- For horse attributes management:
  - Initialize horse instances with user input;
  - Update horse *morale* attribute after every race;
  - Check horse *defeated* status.

Task 1: Basic Horse Racing Game

Three Modules: Team

- ➢For horses’ management in a team:
  - ➢Initialize team instances with horses;
  - ➢Update horse racing order every round with user input.

Task 1: Basic Horse Racing Game

<u>Three Modules: Court</u>

- For game simulation:
    - Initialize all team and horse instances with user input;
    - For every racing round:
        - Match racing horses from two teams in order;
        - Compute racing results and update horse status;
    - Check winning condition for teams and announce the winner if there is one.

Task 1: Basic Horse Racing Game

Requirements

- Implement the game by completing the Perl skeleton in the above three files.
- A Python version of this game is provided. Its OO design is completely the same as the Perl version. You may refer it to better understand the problem.
- Your program should run by executing $perl\ main.pl$. A test case is given as an example to test your code’s correctness.

Outline

- ➢Basic Perl Knowledge Review
  - ➢Static vs. Dynamic Scoping
- ➢Programming Assignment 3 Review
  - ➢Task 1: Basic Horse Racing Game
  - ➢Task 2: Advanced Horse Racing Game
  - ➢Implementation Hints
  - ➢Report & Submission

Task 2: Advanced Horse Racing Game

<u>Problem</u>

- Advanced Horse Racing Problem

  Based on the basic game version, we add some new features as follow:

  - Each horse possesses some coins, which can be utilized to upgrade the properties permanently;
  - Each horse will suffer from ability degradation after each round;
  - Add advanced rules to award/punish horses under several conditions.
- Please refer to Part 2 of Assignment Specification for more details.

Task 2: Advanced Horse Racing Game

Two Modules

- ➢AdvancedHorse.pm

  Inherits the Horse module, define the advanced horse instance.

- ➢AdvancedCourt.pm

  Inherits the Court module, define the advanced game engine and simulate the racing game with new workflow.

Task 2: Advanced Horse Racing Game

Two Modules: AdvancedHorse

- ➢New features including:
  - ➢Manage *coins* for each horse;
  - ➢Upgrade horse attribute(s) with user input if there are enough *coins*;
  - ➢Upgrade horse attribute(s) after every race.

Task 2: Advanced Horse Racing Game

Two Modules: AdvancedCourt

- New features including:
  - For every racing round:
    - In every race, check if any of the advanced rules is satisfied, perform the corresponding action;
    - Prompt the user to upgrade horse properties with coins.

Task 2: Advanced Horse Racing Game

Requirements

- Implement the game by completing the Perl skeleton in the above two files.
- Also complete the implementation of advanced version with Python.
- Your Perl program is still run by executing $perl\ main.pl$. Python program is executed with $python3\ main.py$. A test case for this advanced version is given as an example to test your code's correctness.

Outline

- ➢Basic Perl Knowledge Review
  - ➢Static vs. Dynamic Scoping

- ➢Programming Assignment 3 Review
  - ➢Task 1: Basic Horse Racing Game
  - ➢Task 2: Advanced Horse Racing Game
  - ➢Implementation Hints
  - ➢Report & Submission

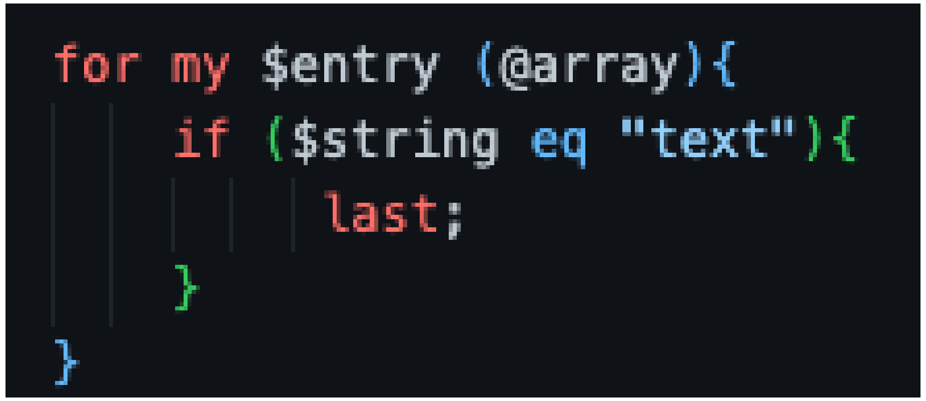
Implementation Hints

- Main Class Usage
  - Follow the instructions in file, uncomment the corresponding lines.

- Initialization of Variables
  - Initialize with default value:
    - Perl: `_round_cnt => 1`
    - Python: `self.round_cnt = 1`
  - Initialize with None or undef:
    - Perl: `_team1 => undef`
    - Python: `self.team1 = None`

Implementation Hints

- Break Out of a Loop
  - Use “last”.



```perl
for my $entry (@array){
        if ($string eq "text"){
            last;
        }
}
```

Implementation Hints

- Code Workflow Abstraction
  - Take Python version as an example:
  - play_game():
    - Invoke input_horses() to obtain horses information;
    - Initialize teams and set horses to teams;
    - Start game, for every iteration (round):
      - Obtain horse racing orders for both teams;
      - Invoke play_one_round(), for every iteration (race):
        - Obtain racing horses and their properties;
        - Compare horse attributes and update morale;
      - Output racing results;
      - Invoke check_winner(), check team winning status.
  - You may refer to the Python code and understand the code before implementing in Perl.

Outline

- Basic Perl Knowledge Review
  - Static vs. Dynamic Scoping

- Programming Assignment 3 Review
  - Task 1: Basic Horse Racing Game
  - Task 2: Advanced Horse Racing Game
  - Implementation Hints
  - Report & Submission

Written Report & Submission

Written Report

- It should be a report within 2 A4 pages, explaining:

1. Where is dynamic scoping used in your Perl code, provide necessary elaborations on its advantages compared with its corresponding code in Python.
2. What is the keyword *local* for, and how it is used in your implementation.

Written Report & Submission

Submission

- Submit the following files:
  - perl_skeleton:
    - main.pl
    - base_version:
      - Court.pm
      - Horse.pm
      - Team.pm
    - advanced_version:
      - AdvancedCourt.pm
      - AdvancedHorse.pm
  - python_skeleton:
    - main.py
    - base_version:
      - Court.py
      - Horse.py
      - Team.py
    - advanced_version:
      - AdvancedCourt.py
      - AdvancedHorse.py
- We strongly recommend you keep the directory structure of the downloaded assignment, work on the skeleton code, compress the whole directory and submit.

Q&A