

CS306 – Introduction to Perl
Fall 2007
Homework Assignment #1
Due: Monday, September 24th, 2007, 4pm

Guidelines for Submitting Homework

All homework should be submitted in the form of a zip file which contains one program for each question. Your zipfile should be named *lastname-hw1.zip* and should contain one text file called *answers.txt* for the written answers, and then files named *hw1p1.pl*, *hw1p2.pl*, *hw1p3.pl*, etc... for the programming assignments. **Make sure your name also appears in all source code files!** Email this zip file to cs306@cis.uab.edu. **When emailing this file email a copy to yourself so you know the email went through ok.**

Homework #1 (200 total points)

Question 1 (15 points). List three ways in which Perl is different from Java.

Question 2 (5 points). What does the `perldoc` say about how the function 'hex' differs from the function 'oct'?

Question 3 (10 points). What is a *keyword*? Give three examples of keywords. Is it possible to have a variable with the same name as a keyword?

Question 4 (10 points). What is a *statement block*? Give an example of a statement block.

Question 5 (15 points). What is *scoping* and how does it relate to statement blocks? Give an example of scoping that shows a two variables which share the same name but have different scope.

Question 6 (15 points). In your own words, explain and illustrate the concept of the default variable (`$_`) in Perl. You should include a code sample other than what appeared on the slides, and find at least one Perl function that uses the default variable that we haven't talked about in class (`print`, `chomp` and `<STDIN>` were covered in class).

Notes on programs: While writing these programs below, remember to comment thoroughly, including your name near the top as well as comments throughout explaining what you are doing in your program.

Also, turn on warnings and use `strict`; in all of your programs. Here's how:

```
#!/usr/bin/perl -w
use strict;
```

The `/usr/bin/perl` part may change depending on your platform; it's the `-w` that is important, as it is what enables warnings.

Read each problem description carefully, and be sure to address everything that is asked for in the problem.

A NOTE ON DEBUGGING: WHEN YOU GET STUCK, PRINT OUT YOUR VARIABLE VALUES AT VARIOUS POINTS IN YOUR PROGRAM. THIS IS OFTEN THE EASIEST WAY TO FIND WHERE THE CODE IS BROKEN.

Program 1: MPH to KPH (15 points)

Write a program that will convert from miles per hour to kilometers per hour. The input from the user is a number representing the MPH, and the output should be the KPH.

Program 2: Guess the secret password (15 points)

Write a program which asks a user to guess a secret password. If they don't get it right in three tries, exit the program. The secret word should be "cheeseburger" (case-sensitive!) to make it easy to test, and it can be hard-coded into your program.

Program 3: Mad Libs (50 points)

Write a program to play the game "Mad Libs". If you've never seen Mad Libs, it works like this... You buy a book that has a series of pre-printed stories in it, with blanks that are to be filled in later. It looks like this:

This Halloween, I tried something different. I went as a detective! Detectives are <adjective> because they solve mysteries. My dad thought it was a good idea for me to take our <animal, singular> Scooter along, because every good detective has a(n) <same animal>, he said.

Things went along smoothly until Scooter saw our neighbor's <animal, singular> and chased it up a(n) tree. I almost dropped my candy, but luckily, Scooter doesn't <verb, present tense> very <adverb>.

We then went to <male friend's name>'s house to meet him and his little sister. I hate Trick or Treating with little sisters. They are no fun! But John's dad said we had to take her along.

Well, it was a good thing I was a detective, because <same male friend's name>'s little sister ran away from us and we were really scared that she was lost. Luckily, Scooter sniffed her out and brought her back to us. She had only gone across the street to see her friends. I'm such a(n) <adjective> detective!

Mad Libs is played with two people. One asks the other for the various words to fill into the story. The second player has no idea what the story is about. If you're lucky, the second player's word choices make the story very funny, and when you then read it out loud, hilarity ensues! (Ok, I realize this game is for elementary school kids, but bear with me... :-)

Your job is to write a Perl version of Mad Libs. The first part of your program will ask the user for a noun, a verb, an adjective, another noun, etc... I will allow you some flexibility in this, but you must ask for at least **eight** words. The second part of your program will insert these words into place in your story and then print the results.

Here is the pattern your program should follow:

```
print "Please enter a noun: ";
my $noun1 = <STDIN>;
.
.
.
print <<EndStory;
```

The \$noun1 jumped over the \$adjective \$noun2.

EndStory

Of course there are a lot more elements to the program – you must be sure you comply with use strict; you must ask for lots more words than I did here, you must clean up your input, and so on. The absolute requirements are that you must use strict, you must ask for at least 8 words, you must clean up your input data, and you must successfully insert the words into your story and print out your story. You may make up the story as you see fit.

Note that I used a Here Doc (p. 20) to make it easier to read and print out a large block of code. I suggest that you do the same.

BONUS (25 points). At the start of the program present a menu that allows the user to pick from between three different story. Each story must have a different set of words that needs to be entered (they can't all take a noun then three verbs, two adjectives and three adverbs, for example), so the program will have to ask the right questions of the users depending on their choice and print out the right story at the end.

Program: Simple Blackjack (50 points)

This program doesn't really play blackjack, but it approximates the dealing of cards in a single blackjack hand. It works as follows:

1. The program deals two “cards” to the user. In our program, a “card” will simply be a random number between 2 and 11. Our program will not have the concept of the Ace being either 1 or 11, to simplify the logic.
2. The program will then enter a loop asking the user if he wants another card. The user will enter 'h' for hit, or 's' for stand.
3. As the user continues to hit, the program keeps track of the user's score. If the score goes over 21, the user busts and loses immediately.
4. If the user score reaches exactly 21, they win immediately.
5. If the user stands below 21, you then simulate the dealer's hand. The rules for this are easy: the dealer gets two cards to start, then the dealer must keep hitting until their total is 18 or higher. So the dealing only stops when the dealer reaches a score between 18-21, or goes over 21 and busts.
6. If both the user and dealer still have valid hands, compare the values. The one closest to 21 wins. If it's a tie, declare a tie.
7. The program only plays one hand.

A round might look like this:

You are dealt a 5. Your score is now 5.
You are dealt a 3. Your score is now 8.
Do you want to hit? h
You are dealt a 10. Your score is now 18.
Do you want to hit? s
The dealer is dealt a 3. Dealer's score is now 3.
The dealer is dealt a 7. Dealer's score is now 10.
The dealer is dealt a 7. Dealer's score is now 17.
You win!

Another round:

You are dealt a 10. Your score is now 10.
You are dealt a 6. Your score is now 16.
Do you want to hit? h
You are dealt a 8. Your score is now 24. You lose.

Another round:

You are dealt a 10. Your score is now 10.
You are dealt an 11. Your score is now 21. You win!

And so on.

Note that there are several differences between this and real blackjack. This program doesn't account for the "soft scoring" option with an Ace. It also does not know that a deck only has 4 of each card. It doesn't allow for splitting a hand or doubling down, and so on.