Exercise 1: Hello World

In pretty much every programming class, the first thing to try is to output the text "Hello World". Basically, this will make sure that we're familiar enough with the environment to do the rest of the exercises in class. It also lets us know that our installation of the programming language (Perl, in this case) works!

Here's our "Hello World" program:

```
#!/usr/bin/perl

# This is our first exercise

print "Hello, world! \n";
```

Make sure the script does what you'd expect when you execute it, then edit the same script to look like this:

```
#!/usr/bin/perl

# This is our first exercise

print "Hello,";
print "world \n";
print 'Hello\nworld\n';
```

Execute this new version of the script. We'll talk more about why the output looks the way it does.

Exercise 2: Math / Formatting Output

Now that we know something about how to do mathematical operations and how to format output in Perl, let's put it to use. Create a script that does some of the things that we've talked about. Try to include as many of these things as you can:

(Hint: the best way to do this is to add a new print statement to the script for each task. In other words, in the end you should have a script containing a separate print statement for each one of the tasks below)

1. Find the result of 23 divided by 3 and output the result
2. Output the above result with only 3 decimal places
3. Output the above result with no decimal places
4. Output the same number in this format: **The result is: *[10 spaces then the number]***
5. Output the text: **So then Fred says, "Umm, can I borrow $5?"**
6. Use escape sequences to capitalize Fred's quote in the above text
7. Output the text: **Today is *[today's date]***
8. Go back and edit the script to output a row of dashes between each of the above tasks

Exercise 3: Scalars / Getting Input from Users

Write a program to calculate annual interest on a bank account. Ask the user for the annual interest rate (as a percentage) and the balance in the account. Store the answers in scalar variables and return the result nicely formatted.

For example:
Please enter the dollar balance in the account: **1000**
Please enter the annual interest percentage: **5**

The total for $1000 at 5% annual interest is $1050.

Try your program with some dollar and cent amounts.

Advanced

If you already know something about working with strings and maybe regular expressions, try enhancing the program by allowing the user to enter their input in different forms (ie: with commas, dollar signs, etc.). You could also try outputting your dollar amounts with commas in the right places. Can you personalize the output so that it addresses the user by their unix login?

Exercise 4: Arrays

Create a script called foods.pl that asks the user for their favorite foods. Ask the user to enter at least 5 foods, each separated by a space (or some other delimiter). Store their answer in a scalar. Split the scalar into an array. Once the array is created, have the script do the following:

- Print the array
- Print the number of elements in the array
- Create an array slice from three elements of the array and print the new array

Advanced

We haven't talked about conditionals like **if** statements yet, but if you already know something about them you can try editing the foods script to exit with an error message if the user doesn't enter at least 5 foods.

We also haven't talked about looping constructs in Perl, but if you know how to use them you can try editing the script so that it repeatedly asks them for their list of foods until they enter a list with at least 5 items. Better than that, of course, would be to let them know how many more they need to add and allow them to type just those missing ones. Try it.

Exercise 5: More Arrays

Here's another exercise that uses arrays. This time we'll get our array data from a file. Since we haven't talked about how to open a file for reading, you can use this shortcut:

```
while ($line = <>)
{
# This is a loop that iterates through each line of our file.
# The filename is specified as an argument to the script when it's run.
# For each iteration of the loop, the $line variable is set to the
# current line being read.
# This happens until the last line of the file has been read.
}
```

In your home directory there is a text file called "passwd". This is the unix password file for the system that we're logged into. It contains information about all of the users that have an account on this host. Each line corresponds to one user and has 7 fields, separated by colons.

Write a script to do these things or anything else you think might be useful:

- Print a list of users, each followed by the location of their home directory. (fields 1 and 6 are for 'login' and 'home directory' respectively)
- Display the total number of accounts on this system

Advanced

- Call the unix 'whoami' program from your script to find the username that you're logged in as. Display that user's home directory and shell (fields 6 and 7 respectively). Hint: use an **if** statement in the while loop to print the information only if the current line corresponds to the user you're interested in.
- Try using the 'limit' parameter of the split function here to see how it works. Give it a value of 6. What happens?
- If you know how to use the 'if' conditional, try printing information only for users in this class. You'll probably need to use regular expression matching or some type of string searching function, right? Or you could just ask the user for a login and display information only for that account.

Exercise 6: Associative Arrays (Hashes)

Part I

We're going to create an associative array to store Cambridge Center for Adult Education course codes and their corresponding course names. For example, this course is called "Perl Programming" and its course code is "PERL".

Write a script that presents the "PERL" course code to the user and asks for its corresponding course title. Store the two pieces of data in an associative array (hash), where the course code is the key and its value is equal to the course description. Using the hash that you created, print a line like this:

```
The course description for course code is: course_description
```

Part II

Ask the user for a key/value pair. Store their answer in a scalar variable. Find a way to take the key/value pair that they gave you and put it in an associative array.

Ask them a few more times (you can just copy/paste for now - we'll talk about a way to do things repeatedly using loops in a couple of weeks), then try some of these things:

- Print the name for a specific course code
- Print the names and course codes for all of the courses in the hash
- Ask the user for a course code and return the name of the course they've entered

Oh yeah, here are some course codes and names for you to work with, in case you're interested:

ABSP	Abstract Painting
ADMO	Advanced Printmaking
ANAC	Acting I: Anyone Can Act
WSEM	Fiction Writing Seminar
ZNWR	Zen Writing

Advanced

- Once the user has entered the key/value pairs for a bunch of courses. Ask them for a course code and let them know whether or not it exist()'s.
- Try to figure out how to print the contents of the hash in alphabetical order by course code (ie: sorted by key). Hint: the easiest way involves using the **foreach**, which we'll see next week

Exercise 7: Conditional Statements

This is a pretty short exercise just to get you used to if statements. Write a program to take a temperature value (in Farenheit) as input from the user and print a corresponding phrase to the screen depending on the temperature entered. Here are the phrases to use:

- If the temperature is not between -50 and 200 degrees, let the user know that they've probably entered the wrong temperature
- If the temperature is above 100 degrees and less than 200, output the phrase "super hot"
- If the temperature is above 70 degrees but below 100, output the phrase "pretty hot"
- If the temperature is above 50 degrees but below 70, output the phrase "kinda chilly"
- If the temperature is above 0 degrees but below 50, output the phrase "pretty cold"
- If the temperature is above -10 degrees but below 0, output the phrase "really cold"

Remember that the order and manner that I've worded those conditions may or may not be the way you want to write your Perl code. Imagine that your boss comes to you with a set of requirements for a script. Would you necessarily write your code to emulate her description verbatim? Probably not. Even after you get this script working, take another look at it to see if there's a way to make it more efficient (eg: try to minimize the number of statements that are executed).

Advanced

You can also test your newfound knowledge of the if statement by going back to exercises 4, 5, and 6 and doing the first task in each "Advanced" section.

Exercise 8: Loops

Okay, now that we've learned how to use loops, we've got enough concepts down to take on a slightly more involved project. It's not necessarily harder but it'll take more code.

Write a script called "grading.pl" that will take a set of grades as input from the user and output their average. The program should:

1. Ask the user how many grades he or she will enter
2. Use a loop to total up the grades
3. If an invalid grade is entered, the script should tell the user, and ask them to enter another grade
4. Print the average of the grades
5. Ask the user if they want to enter another set of grades. If the answer is yes then the whole process should start again. Otherwise, the script should end with a nice message.

Here's what the whole thing might look like when it's executed:

```
$ grading.pl

How many grades are you going to enter? 4
Please enter your 4 grades, separated by spaces: 44 38 78 83
THE AVERAGE GRADE IN THAT SET WAS: 60.75 %

Do you want to enter another set of grades? yes

How many grades are you going to enter? 2
Please enter your 2 grades, separated by spaces: 190 -27
190 is an invalid grade!
Enter a new grade: 50
-27 is an invalid grade!
Enter a new grade: 89
THE AVERAGE GRADE IN THAT SET WAS: 69.5 %

Do you want to enter another set of grades? no

Thanks, goodbye.
```

Advanced

Modify the script so that it doesn't have to ask how many grades will be entered. Use a loop that keeps asking until the user signals (however you'd like - maybe a blank line?) that they're through entering the current set of grades. This might be a better way since you can show them the error message as soon as they enter an invalid grade, rather than waiting until they've entered them all to check as I did in the above sample output.

Exercise 9: File I/O

In this exercise, you'll open a file for reading and write some of its lines to a separate file. The data file is called employees.txt and contains the following lines:

```
001:Jim Jones:45000
002:Jerry Holden:43000
003:Larry Walden:56200
004:Frank Williams:54000
005:Mary Miret:60000
006:Ed Tran:30000
009:Dennis Bird:44000
011:Richard Sanders:42750
```

The file contains employee information - one line per employee, with 3 colon-delimited fields per line. The fields are "Employee ID", "Employee Name", and "Employee Salary". Here's what your script should do:

- Open a filehandle for reading employees.txt
- Read the rest of the file and write any lines with a salary of more than $50,000 to a separate file called highrollers.txt

That's it. Don't forget to close any filehandles that you open!

Advanced

Allow the user to add new employees to the employees.txt file. Ask the user for the 3 employee details and add the line to the end of the file. Make a loop so the user can enter as many employees as they'd like.

Exercise 10: Regular Expressions / Pattern Matching

In this exercise, you'll use a while loop to iterate through a data file and print only lines that match a certain pattern. The data file is called books.txt and contains a list of books.

Try each of the following tasks in turn. The best way to do this is to comment out the previous task as you move on to the next one. Here they are:

- First of all, skip any blank or commented lines
- Print all lines containing the string "ter"
- Print the lines for author "Saga"
- It's been deemed that the word "Hell" is unsuitable for our script. Print all lines. Any instances of the word "Hell" should be changed to "H*ll"
- Print the lines for ISBN's that do NOT contain the number 6
- Print the lines for books that were published after 1995
- Print the first word of each book title. One per line

Advanced

Here's something else to try. Iterate through the passwd file we used in Exercise 5 and output the lines that relate to users in this class. If you're really ambitious, you can try to output the username and the NAME of the primary group that they belong to. The group ID to group name mapping is kept in the `/etc/group` file.

Data File Contents

The contents of books.txt are shown below. Be aware of where the lines really end.

```
# this data file is pipe ('|') delimited and contains these fields:
# isbn, title, author(s), publisher, pages, publication date

0061095508|Miss America|Howard Stern|Harper Mass Market Paperbacks|592|11/1996
0345410084|Hell's Angels: A Strange and Terrible Saga|Hunter S. ↲
 Thompson|Ballantine Books|273|09/1996
0961392126|Visual Explanations|Edward R. Tufte|Graphics Press|156|03/1997
030680817X|Stairway to Hell: The 500 Best Heavy Metal Albums in the ↲
 Universe|Chuck Eddy|Da Capo Pr|288|04/1998
0802713661|Brunelleschi's Dome : How a Renaissance Genius Reinvented ↲
 Architecture|Ross King|Walker & Co|194|10/2000
0393320928|What Do You Care What Other People Think?: Further Adventures of a ↲
 Curious Character|Ralph Leighton, Richard Phillips Feynman|W.W. Norton & ↲
 Company|256|01/2001
0387979352|Light and Color in the Outdoors|M. G. J. Minnaert|Springer ↲
 Verlag|417|07/1993
0962094404|Banned in D C : Photos and Anecdotes from the DC Punk ↲
 Underground|Cynthia Connolly, Leslie Clague, Sharon Cheslow|Sun Dog ↲
 Propaganda|176|11/1988
0880486759|Dsm-IV Casebook : A Learning Companion to the Diagnostic and ↲
 Statistical Manual of Mental Disorders|Robert L. Spitzer, Miriam Gibbon, ↲
 Andrew E. Skodol, Michael B. First|Amer Psychiatric Pr|576|07/1994
4770019483|Confessions of a Yakuza : A Life in Japan's Underworld|Funichi ↲
 Saga|Kodansha International||06/1995
```

Exercise 11: Final Exercise

This exercise will use everything that you've learned to create a script that will go through the contents of a webpage and save selected information to a file. Here's what you'll need to do:

- First, you need the webpage. I'll talk, in class, about what you'd need to consider if you wanted to integrate this part into your script. For now, let's just get it using unix commands. Try `lynx -source http://boston.craigslist.org/sss/ > myfile.html` If that doesn't work, we can use `GET http://boston.craigslist.org/sss/ > myfile.html` Again, I'll talk more about what's happening here in class. By the way, for this exercise use the URL shown here - I chose a classifieds page from craigslist because the HTML is very clean and simple and it'll make the exercise easier for you. You can, of course, do this with other web pages after you get the hang of pattern matching.
- Once you have the webpage, you'll write a script that will read through each line of the HTML file and write only the name of each of the forsale items to a file called forsale.txt

Advanced

The exercise above is called "scraping" and is wrought with potential problems. What happens if craigslist changes the format of their HTML pages? Your script is likely to break – that's what happens! Wouldn't it be great if there was an alternate version of this web page that was designed to be read by scripts like ours instead of a human sitting at a web browser? Look into RSS and explain how it could help us here.