Perl Operators

Arithmetic Operators

Operator/Operands	Purpose
$a + $b	Add
$a - $b	Subtract
$a * $b	Multiply
$a / $b	Divide
$a % $b	Modulus (remainder)
$a ** $b	Exponent ($a to the $b power)
$a++	Auto-increment ($a plus 1)
$a--	Auto-decrement ($a minus 1)

String Operators

Operator/Operands	Purpose
$string1 . $string2	Concatenate
$string x $num	Repeat $string, $num times

Assignment Operators

Operator/Operands	Purpose
$a = 5	Set the value of $a to 5
$a += 5	Add 5 to the current value of $a
$a -= 5	Subtract 5 from the current value of $a
$string1 .= $string2	Set the value of $string1 to its current string value with the value of $string2 added to the end of it

Numeric Relational Operators

Operator/Operands	Evaluates to true if
$a == $b	$a is equal to $b
$a != $b	$a is not equal to $b
$a > $b	$a is greater than $b
$a >= $b	$a is greater than or equal to $b
$a < $b	$a is less than $b
$a <= $b	$a is less than or equal to $b

String Relational Operators

Operator/Operands	Evaluates to true if
$string1 eq $string2	$string1 is equal to $string2
$string1 ne $string2	$string1 is not equal to $string2

Logical Operators

Operator/Operands	Alternative	Purpose
$a && $b	and	Returns $b, if $a and $b are both true
$a || $b	or	Returns $a, if either $a or $b is true
! $a	not	Returns true if $a is false

Formatting Output

Using the printf function

printf FORMAT, LIST

FORMAT is a string that can contain format specifications in the form: **%-m.nx**

m is the amount of space that the string or number should take up, with an optional dash in front of it to specify that it should be left justified. This part of the format string is optional.
n the number of decimal places, if applicable
x the data type to display - see below

Code	Meaning
c	Character
d	Decimal integer
f	Floating point number
e	Floating point number in scientific notation
s	String
x	Hex number
o	Octal number

Using Escape Sequences in String Literals

Escape Sequence	Represents
\t	tab
\n	newline
\r	return
\f	form feed
\b	backspace
\a	alarm
\e	escape
\033	octal character
\xff	hex character
\c[	control character
\l	next character lower case
\u	next character upper case
\L	following characters are lowercase until \E is encountered
\U	following characters are uppercase until \E is encountered
\E	ends \L or \U

Executing Perl Scripts

Okay, so here's a quick review of what we've already done in class to execute our Perl scripts from Exercise 1:

One sure-fire way to run your script is to call the Perl interpreter and give it the name of your script file: **$ perl exercise1.pl**

(You didn't have to name your file exercise1.pl, but it's a good convention to put .pl at the end of your script filenames.)

If you'd like to be able to run your Perl script without typing "perl" before the name of the script file, then there are a couple of other things that need to be done:

1. Make sure your script identifies itself to Unix as a Perl script by having this as its first line: **#!/usr/bin/perl**
2. Turn on the Unix "execute" permission for your script file by typing this command at the Unix prompt: **$ chmod +x exercise1.pl**
3. Run your script by typing its filename

Perl Web Resources

This list of Perl resources on the Internet is roughly in order of importance. Sort of.

http://www.perl.com/ is the main source for Perl information. It's run by O'Reilly, who also happen to publish a lot of great computer books. The "Learning Perl" book that I recommended in class is one of theirs. Their company website is at http://www.ora.com/ and you can recognize books that they've published by the animals on the covers.

http://perldoc.perl.org/ has all of the Perl documentation online.

During class I suggested that you might want to install Perl on your home computer. If you have Mac OS X, it's already there. You can do everything that we do in class by using the Terminal program in the Utilities folder (It's in Applications). TextMate is a great programmer's editor for Mac OS users. It's at http://www.macromates.com/. Or you can use VI since it's already a part of the OS.

If you're using Windows, you can get Perl from ActiveState at http://www.activestate.com/. ActiveState is a commercial company that offers their own versions of Perl. They have versions for Windows and Unix. Don't worry it's still free and it's basically the same as the version that we're using in class. They do add a couple features that make installation easier and if you're downloading their Windows version, it comes with a bunch of cool Windows-specific features.

http://www.perl.org/ is the home of the Perl Mongers, which is an organization of Perl advocates. When you go to this site, you should sign up for some of the mailing lists - specifically the "beginners" mailing list that they recently started. There's a Boston chapter of the Perl Mongers, called Boston.pm. Their website is at http://boston.pm.org/ and they have meetings on the second Tuesday of each month at the Boston.com downtown.

http://learn.perl.org/ is the site for people learning Perl. (That's you.)

It's probably a little soon to start looking for a Perl job, but I know a few people asked what kinds of jobs require Perl experience. Look at http://jobs.perl.org/. You can also sign up for the mailing list so you get the job listings as they're posted. Of course, searching for Perl in the craigslist.org jobs section works.

The Perl Journal is a good print magazine that comes out 4 times a year. Try to get your boss to pay for a subscription. There's some good stuff in here. Anyway, they have a website at http://www.tpj.com/. They have information about the Obfuscated Perl Contest that I mentioned in class. The entries in the Perl Poetry Contest are pretty entertaining to look at, too.

I think http://www.perlmonks.com/ is more for advanced Perl programmers, but you can bookmark it again and look at it in a couple weeks when you're a Perl expert.

If you're interested in finding out more about Perl6, look at http://dev.perl.org/

Common Perl Mistakes

As we continue in the course, we'll add more common mistakes to this list. If you want to see a big list of common mistakes right now, you can look at the Perl Traps for the Unwary section of the Perl documentation by typing `perldoc perltrap` at the shell prompt. In the meantime, here are some that we've run into already in class and some that we'll see soon enough.

- Make sure you use the "-w" switch to turn on warnings until you get your script working properly.
- If you get a "Permission Denied" message when trying to execute your script, it's probably because the Unix "execute" permission isn't turned on for your script file. See our **Executing Scripts** handout for more info.
- If you're trying to execute your script and you get a message saying the file isn't found, it's probably because the first line of your script isn't correctly pointing to the perl binary. Again, see our **Executing Scripts** handout.
- Use '==' for comparing numbers. Use 'eq' for comparing strings. See our **Operators** handout.
- Make sure you're not accidentally using the '=' assignment operator when you mean to use the '==' comparison operator. An expression like **$x = 5** will always evaluate to true. **$x == 5** is probably what you wanted.
- **Elsif** only has one 'e'.
- If you try to write to a file that has been opened for reading-only, you will get a message saying that you've tried to "write to a closed filehandle." That's only if you have warnings turned on though! Otherwise, the script will execute without any errors or warnings but the file that you were writing to won't have your additions.
- If you're trying to write a CGI script, you should know that the first thing it has to output is at least one HTTP header. That header describes the type of content that follows and could look like: **print "Content-type: text/html\n\n";** Note the capitalization and the extra returns at the end of the string. Also note that you may need to turn off warnings on CGI scripts since a warning might get output before your header, which will cause a server error.

Perl File I/O

Opening Files

Syntax	Result
open FH, "filename";	filename is opened for reading only
open FH, "<filename";	same as above
open FH, ">filename";	filename is opened for writing only. The file is TRUNCATED first!
open FH, ">>filename";	filename is opened for appending
open FH, "+>filename";	filename is opened for writing and reading. The file is TRUNCATED first!
open FH, "+<filename";	filename is opened for reading and writing. (Cursor is at the beginning of the file)
open FH, "+>>filename";	filename is opened for reading and appending. (Cursor is at the end of the file)

After any of the above calls to the **open** function, the filehandle "FH" is used to refer to the file named "filename".

Here are some notes about opening files:

- You can call your filehandle anything you want but it should be in all capital letters with no punctuation
- A filename without any path information assumes that the file is in the current directory. If the file is in another directory, you'd have to specify the full path to the file (eg: "/etc/passwd").
- Don't forget to **close** the filehandle after you're done reading and/or writing it.
- As a beginner, you probably won't use the last three versions shown in the table above. If you're interested in using them, you'll also need to know about the **seek**, **tell**, and **read** functions.
- If you're using Windows and you're opening a binary file, you will have to call the **binmode** function after calling **open**.

Catching Errors

File I/O functions like **open** return false if they fail. You can use this behavior to handle errors. For example:

- open FH, "filename" || die("Error opening file: $!\n");
- unless (open FH, "filename") { die("Error opening file: $!\n") }

die takes a string as an argument and prints that string to STDERR. It also causes the script to halt at that point. If the string doesn't end with "\n", the name and line number of your script will also be displayed to the user. $! is a special variable that contains the most recent operating system error (eg: "file not found").

Perl File I/O

(continued)

Reading from a Filehandle

- **@array = <FH>** will "slurp" the entire file into the array. Each line of the file will be a separate element in the array. This can be a bad idea if the file is large since you're basically making a copy of the entire file in memory (in the @array variable).
- **while ($line = <FH>) { ... }** will read each line of the file into **$line** in turn. Only one line of the file is in memory at a time, which makes this a better option in most cases.

Writing to a Filehandle

- **print FH "text\n";** will write the string to the **FH** filehandle. The file must be open for writing or you will get a "tried to write to closed filehandle" message from perl (you do have warnings turned on don't you?). Note that there's no comma after the filehandle name.
- If you're doing a lot of writing to the same filehandle, you may want to look into the **select** function.

File Test Operators

You can check on various attributes of a file by using the perl file test operators. They are identical to those used in Unix shell scripts and they allow you to find out things like whether or not a file exists, when it was last accessed, and what its permissions are. Check out the file test operators section of the perl documentation by typing `perldoc –f –X` at the prompt. Note the stuff about the **stat** function at the bottom. **stat** is a function that returns a list of attributes about a given file.

Perl Regular Expressions Reference

Here's a partial list of some useful things you can use in a Perl regular expression.

Metacharacters

Characters	Function
.	Matches any character except newline
[a-z0-9]	(Character class). Matches any single character in the set
[^a-z0-9]	Matches any single character not in the set
\d	Matches a single digit 0-9
\D	Matches a single non-digit
\w	Matches a single "word" character (alphanumeric or "_")
\W	Matches a single non-word character
\s	Matches a whitespace character (eg: space or tab)
\S	Matches a non-whitespace character
\n	Matches a newline
\t	Matches a tab
\0	Matches a null character
\b	Matches a word boundary (when not inside a character class)
^	Matches the beginning of the string (when not inside a character class)
$	Matches the end of the string

Quantifiers

Characters	Function
x?	Matches 0 or 1 x's
x*	Matches 0 or more x's
x+	Matches 1 or more x's
x{m,n}	Matches at least m x's, and no more than n x's
x|y	(Alternation). matches x OR y

Backreferences

Use parentheses in a regular expression to be able to backreference that part of the match. $1, $2, and so on are used to refer back to the part of the match in parentheses.