Why do you use Perl?
Perl is a powerful free interpreter.
Perl is portable, flexible and easy to learn.

How do I set environment variables in Perl programs?
you can just do something like this:
$path = $ENV{'PATH'};
As you may remember, "%ENV" is a special hash in Perl that contains the value of all your environment variables.
Because %ENV is a hash, you can set environment variables just as you'd set the value of any Perl hash variable. Here's how you can set your PATH variable to make sure the following four directories are in your path::
$ENV{'PATH'} = '/bin:/usr/bin:/usr/local/bin:/home/yourname/bin';

Which of these is a difference between C++ and Perl?
Perl can have objects whose data cannot be accessed outside its class, but C++ cannot.
Perl can use closures with unreachable private data as objects, and C++ doesn't support closures. Furthermore, C++ does support pointer arithmetic via `int *ip = (int*)&object', allowing you do look all over the object. Perl doesn't have pointer arithmetic. It also doesn't allow `#define private public' to change access rights to foreign objects. On the other hand, once you start poking around in /dev/mem, no one is safe.

How to open and read data files with Perl
Data files are opened in Perl using the open() function. When you open a data file, all you have to do is specify (a) a file handle and (b) the name of the file you want to read from.
As an example, suppose you need to read some data from a file named "checkbook.txt". Here's a simple open statement that opens the checkbook file for read access: open (CHECKBOOK, "checkbook.txt");

In this example, the name "CHECKBOOK" is the file handle that you'll use later when reading from the checkbook.txt data file. Any time you want to read data from the checkbook file, just use the file handle named "CHECKBOOK".
Now that we've opened the checkbook file, we'd like to be able to read what's in it. Here's how to read one line of data from the checkbook file:

```
$record = < CHECKBOOK > ;
```

After this statement is executed, the variable $record contains the contents of the first line of the checkbook file. The "<>" symbol is called the line reading operator.
To print every record of information from the checkbook file

```
open (CHECKBOOK, "checkbook.txt") || die "couldn't open the file!";
while ($record = < CHECKBOOK >) {
print $record;
}
close(CHECKBOOK);
```

How do I do fill_in_the_blank for each file in a directory?
Here's code that just prints a listing of every file in the current directory:

```
#!/usr/bin/perl -w
opendir(DIR, ".");
@files = readdir(DIR);
closedir(DIR);
foreach $file (@files) {
print "$file\n";
}
```

How do I do fill_in_the_blank for each file in a directory?

Here's code that just prints a listing of every file in the current directory:

```
#!/usr/bin/perl -w
opendir(DIR, ".");
@files = readdir(DIR);
closedir(DIR);
foreach $file (@files) {
print "$file\n";
}
```

How do I generate a list of all .html files in a directory?
Here's a snippet of code that just prints a listing of every file in the current directory that ends with the extension .html:

```
#!/usr/bin/perl -w
opendir(DIR, ".");
@files = grep(/\.html$/,readdir(DIR));
closedir(DIR);
foreach $file (@files) {
print "$file\n";
```

}

What is Perl one-liner?
There are two ways a Perl script can be run:
--from a command line, called one-liner, that means you type and execute immediately on the command line. You'll need the -e option to start like "C:\ %gt perl -e "print \"Hello\";". One-liner doesn't mean one Perl statement. One-liner may contain many statements in one line.
--from a script file, called Perl program.

Assuming both a local($var) and a my($var) exist, what's the difference between ${var} and ${"var"}?
${var} is the lexical variable $var, and ${"var"} is the dynamic variable $var.
Note that because the second is a symbol table lookup, it is disallowed under `use strict "refs"'. The words global, local, package, symbol table, and dynamic all refer to the kind of variables that local() affects, whereas the other sort, those governed by my(), are variously knows as private, lexical, or scoped variable.

What happens when you return a reference to a private variable?
Perl keeps track of your variables, whether dynamic or otherwise, and doesn't free things before you're done using them.

How to turn on Perl warnings? Why is that important?
Perl is very forgiving of strange and sometimes wrong code, which can mean hours spent searching for bugs and weird results. Turning on warnings helps uncover common mistakes and strange places and save a lot of debugging time in the long run. There are various ways of turning on Perl warnings:
For Perl one-liner, use -w option on the command line.
On Unix or Windows, use the -w option in the shebang line (The first # line in the script). Note: Windows Perl interpreter may not require it.
For other systems, choose compiler warnings, or check compiler documentation.

What are scalar data and scalar variables?
Perl has a flexible concept of data types. Scalar means a single thing, like a number or string. So the Java concept of int, float, double and string equals to Perl\'s scalar in concept and the numbers and strings are exchangeable. Scalar variable is a Perl variable that is used to store scalar data. It uses a dollar sign $ and followed by one or more alphanumeric characters or underscores. It is case sensitive.

Why should I use the -w argument with my Perl programs?
Many Perl developers use the -w option of the interpreter, especially during the development stages of an application. This warning option turns on many warning messages that can help you understand and debug your applications.
To use this option on Unix systems, just include it on the first line of the program, like this:
```
#!/usr/bin/perl -w
```
If you develop Perl apps on a DOS/Windows computer, and you're creating a program named myApp.pl, you can turn on the warning messages when you run your program like this:
```
perl -w myApp.pl
```

Assuming $_ contains HTML, which of the following substitutions will remove all tags in it?
1.s/<.*>//g;
2.s/<.*?>//gs;
3.s/<\/?[A-Z]\w*(?:\s+[A-Z]\w*(?:\s*=\s*(?:(["']).*?\1|[\w-.]+))?)*\s*>//gsix;

You can't do that.
If it weren't for HTML comments, improperly formatted HTML, and tags with interesting data like < SCRIPT >, you could do this. Alas, you cannot. It takes a lot more smarts, and quite frankly, a real parser.

I want users send data by formmail but when they send nothing or call it from web site they will see error.
codes in PHP like this:

```
if (isset($HTTP_POST_VARS)){
..........
}
else{
echo ("error lalalalal")
}
```

How it will look in perl?

In php it will be like

```
if (isset($HTTP_POST_VARS)){
....
}
```

In perl, tried this.

```
if ($ENV{'REQUEST_METHOD'} eq 'POST'){
.....
}
```

What is the output of the following Perl program?

```
1 $p1 = "prog1.java";
2 $p1 =~ s/(.*)\.java/$1.cpp/;
3 print "$p1\n";
```

```
prog1.cpp
```

Why aren't Perl's patterns regular expressions?

Because Perl patterns have backreferences.
A regular expression by definition must be able to determine the next state in the finite automaton without requiring any extra memory to keep around previous state. A pattern /([ab]+)c\1/ requires the state machine to remember old states, and thus disqualifies such patterns as being regular expressions in the classic sense of the term.

What does Perl do if you try to exploit the execve(2) race involving setuid scripts?

Sends mail to root and exits.
It has been said that all programs advance to the point of being able to automatically read mail. While not quite at that point (well, without having a module loaded), Perl does at least automatically send it.

How do I do < fill-in-the-blank > for each element in a hash?
Here's a simple technique to process each element in a hash:

```
#!/usr/bin/perl -w

%days = (
'Sun' =>'Sunday',
'Mon' => 'Monday',
'Tue' => 'Tuesday',
'Wed' => 'Wednesday',
'Thu' => 'Thursday',
'Fri' => 'Friday',
'Sat' => 'Saturday' );

foreach $key (sort keys %days) {
print "The long name for $key is $days{$key}.\n";
}
```

How do I sort a hash by the hash key?
Suppose we have a class of five students.
Their names are kim, al, rocky, chrisy, and jane.

Here's a test program that prints the contents
of the grades hash, sorted by student name:

```
#!/usr/bin/perl -w

%grades = (
kim => 96,
al => 63,
rocky => 87,
chrisy => 96,
jane => 79,
);

print "\n\tGRADES SORTED BY STUDENT NAME:\n";
foreach $key (sort (keys(%grades))) {
print "\t\t$key \t\t$grades{$key}\n";
}
```

The output of this program looks like this:

```
GRADES SORTED BY STUDENT NAME:
al 63
chrisy 96
jane 79
kim 96
rocky 87

}
```

How do you print out the next line from a filehandle with all its bytes reversed?

```
print scalar reverse scalar <FH>
```

Surprisingly enough, you have to put both the reverse and the <FH> into scalar context separately for this to work.

How do I send e-mail from a Perl/CGI program on a Unix system?

Sending e-mail from a Perl/CGI program on a Unix computer system is usually pretty simple. Most Perl programs directly invoke the Unix sendmail program. We'll go through a quick example here. Assuming that you've already have e-mail information you need, such as the send-to address and subject, you can use these next steps to generate and send the e-mail message:

```
# the rest of your program is up here ...
open(MAIL, "|/usr/lib/sendmail -t");
print MAIL "To: $sendToAddress\n";
print MAIL "From: $myEmailAddress\n";
print MAIL "Subject: $subject\n";
print MAIL "This is the message body.\n";
print MAIL "Put your message here in the body.\n";
close (MAIL);
```

How to read from a pipeline with Perl

Example 1:

To run the date command from a Perl program, and read the output
of the command, all you need are a few lines of code like this:

```
open(DATE, "date|");
$theDate = <DATE>;
close(DATE);
```

The open() function runs the external date command, then opens
a file handle DATE to the output of the date command.

Next, the output of the date command is read into
the variable $theDate through the file handle DATE.

Example 2:

The following code runs the "ps -f" command, and reads the output:

```
open(PS_F, "ps -f|");
while (<PS_F>) {
($uid,$pid,$ppid,$restOfLine) = split;
# do whatever I want with the variables here ...
}
close(PS_F);
```

Why is it hard to call this function: sub y { "because" }
Because y is a kind of quoting operator.
The y/// operator is the sed-savvy synonym for tr///. That means y(3) would be like tr(), which would be looking for a second string, as in tr/a-z/A-Z/, tr(a-z)(A-Z), or tr[a-z][A-Z].

What does `$result = f() .. g()' really return?
False so long as f() returns false, after which it returns true until g() returns true, and then starts the cycle again.
This is scalar not list context, so we have the bistable flip-flop range operator famous in parsing of mail messages, as in `$in_body = /^$/ .. eof()'. Except for the first time f() returns true, g() is entirely ignored, and f() will be ignored while g() later when g() is evaluated. Double dot is the inclusive range operator, f() and g() will both be evaluated on the same record. If you don't want that to happen, the exclusive range operator, triple dots, can be used instead. For extra credit, describe this:
$bingo = (a() .. b()) ... (c() .. d());

Why does Perl not have overloaded functions?
Because you can inspect the argument count, return context, and object types all by yourself.
In Perl, the number of arguments is trivially available to a function via the scalar sense of @_, the return context via wantarray(), and the types of the arguments via ref() if they're references and simple pattern matching like /^\d+$/ otherwise. In languages like C++ where you can't do this, you simply must resort to overloading of functions.

What does read() return at end of file?
0
A defined (but false) 0 value is the proper indication of the end of file for read() and sysread().

What does `new $cur->{LINK}' do? (Assume the current package has no new() function of its own.)
$cur->new()->{LINK}
The indirect object syntax only has a single token lookahead. That means if new() is a method, it only grabs the very next token, not the entire following expression.
This is why `new $obj[23] arg' does't work, as well as why `print $fh[23] "stuff\n"' does't work. Mixing notations between the OO and IO notations is perilous. If you always use arrow syntax for method calls, and nothing else, you'll not be surprised.

How do I sort a hash by the hash value?
Here's a program that prints the contents

of the grades hash, sorted numerically by the hash value:

```
#!/usr/bin/perl -w

# Help sort a hash by the hash 'value', not the 'key'.
to highest).
sub hashValueAscendingNum {
$grades{$a} <=> $grades{$b};
}



# Help sort a hash by the hash 'value', not the 'key'.
# Values are returned in descending numeric order
# (highest to lowest).
sub hashValueDescendingNum {
$grades{$b} <=> $grades{$a};
}


%grades = (
student1 => 90,
student2 => 75,
student3 => 96,
student4 => 55,
student5 => 76,
);

print "\n\tGRADES IN ASCENDING NUMERIC ORDER:\n";
foreach $key (sort hashValueAscendingNum (keys(%grades))) {
print "\t\t$grades{$key} \t\t $key\n";
}

print "\n\tGRADES IN DESCENDING NUMERIC ORDER:\n";
foreach $key (sort hashValueDescendingNum (keys(%grades))) {
print "\t\t$grades{$key} \t\t $key\n";
}
```

How to read file into hash array ?

```
open(IN, "<name_file")
or die "Couldn't open file for processing: $!";
while (<IN>) {
chomp;
$hash_table{$_} = 0;
}
```

close IN;

print "$_ = $hash_table{$_}\n" foreach keys %hash_table;

How do you find the length of an array?
$@array

What value is returned by a lone `return;' statement?
The undefined value in scalar context, and the empty list value () in list context.
This way functions that wish to return failure can just use a simple return without worrying about the context in which they were called.

What's the difference between /^Foo/s and /^Foo/?
The second would match Foo other than at the start of the record if $* were set.
The deprecated $* flag does double duty, filling the roles of both /s and /m. By using /s, you suppress any settings of that spooky variable, and force your carets and dollars to match only at the ends of the string and not at ends of line as well -- just as they would if $* weren't set at all.

Does Perl have reference type?
Yes. Perl can make a scalar or hash type reference by using backslash operator.
For example
$str = "here we go"; # a scalar variable
$strref = \$str; # a reference to a scalar

@array = (1..10); # an array
$arrayref = \@array; # a reference to an array
Note that the reference itself is a scalar.

How to dereference a reference?
There are a number of ways to dereference a reference.
Using two dollar signs to dereference a scalar.
$original = $$strref;
Using @ sign to dereference an array.
@list = @$arrayref;
Similar for hashes.

What does length(%HASH) produce if you have thirty-seven random keys in a newly created hash?
5
length() is a built-in prototyped as sub length($), and a scalar prototype silently changes aggregates into radically different forms. The scalar sense of a hash is false (0) if it's empty, otherwise it's a string representing the fullness of the buckets, like "18/32" or "39/64". The length of that string is likely to be 5. Likewise, `length(@a)' would be 2 if there were 37 elements in @a.

If EXPR is an arbitrary expression, what is the difference between $Foo::{EXPR} and *{"Foo::".EXPR}?

The second is disallowed under `use strict "refs"'.
Dereferencing a string with *{"STR"} is disallowed under the refs stricture, although *{STR} would not be. This is similar in spirit to the way ${"STR"} is always the symbol table variable, while ${STR} may be the lexical variable. If it's not a bareword, you're playing with the symbol table in a particular dynamic fashion.

How do I do < fill-in-the-blank > for each element in an array?

```
#!/usr/bin/perl -w
@homeRunHitters = ('McGwire', 'Sosa', 'Maris', 'Ruth');
foreach (@homeRunHitters) {
print "$_ hit a lot of home runs in one year\n";
}
```

How do I replace every <TAB> character in a file with a comma?

```
perl -pi.bak -e 's/\t/,/g' myfile.txt
```

What is the easiest way to download the contents of a URL with Perl?

Once you have the libwww-perl library, LWP.pm installed, the code is this:

```
#!/usr/bin/perl
use LWP::Simple;
$url = get 'http://www.websitename.com/';
```

How to concatenate strings with Perl?

Method #1 - using Perl's dot operator:

```
$name = 'checkbook';
$filename = "/tmp/" . $name . ".tmp";
```

Method #2 - using Perl's join function

```
$name = "checkbook";
$filename = join "", "/tmp/", $name, ".tmp";
```

Method #3 - usual way of concatenating strings

```
$filename = "/tmp/${name}.tmp";
```

How do I read command-line arguments with Perl?

With Perl, command-line arguments are stored in the array named @ARGV.
$ARGV[0] contains the first argument, $ARGV[1] contains the second argument, etc.
$#ARGV is the subscript of the last element of the @ARGV array, so the number of arguments on the command line is $#ARGV + 1.
Here's a simple program:

```
#!/usr/bin/perl
$numArgs = $#ARGV + 1;
```

```
print "thanks, you gave me $numArgs command-line arguments.\n";
foreach $argnum (0 .. $#ARGV) {
print "$ARGV[$argnum]\n";
}
```

When would `local $_' in a function ruin your day?
When your caller was in the middle for a while(m//g) loop
The /g state on a global variable is not protected by running local on it. That'll teach you to stop using locals. Too bad $_ can't be the target of a my() -- yet.

What happens to objects lost in "unreachable" memory, such as the object returned by Ob->new() in `{ my $ap; $ap = [Ob->new(), \$ap]; }' ?
Their destructors are called when that interpreter thread shuts down.
When the interpreter exits, it first does an exhaustive search looking for anything that it allocated. This allows Perl to be used in embedded and multithreaded applications safely, and furthermore guarantees correctness of object code.

Assume that $ref refers to a scalar, an array, a hash or to some nested data structure. Explain the following statements:
$$ref; # returns a scalar
$$ref[0]; # returns the first element of that array
$ref- > [0]; # returns the first element of that array
@$ref; # returns the contents of that array, or number of elements, in scalar context
$&$ref; # returns the last index in that array
$ref- > [0][5]; # returns the sixth element in the first row
@{$ref- > {key}} # returns the contents of the array that is the value of the key "key"

How do you match one letter in the current locale?
/[^\W_\d]/
We don't have full POSIX regexps, so you can't get at the isalpha() <ctype.h> macro save indirectly. You ask for one byte which is neither a non-alphanumunder, nor an under, nor a numeric. That leaves just the alphas, which is what you want.

How do I print the entire contents of an array with Perl?
To answer this question, we first need a sample array. Let's assume that you have an array that contains the name of baseball teams, like this:
@teams = ('cubs', 'reds', 'yankees', 'dodgers');
If you just want to print the array with the array members separated by blank spaces, you can just print the array like this:
@teams = ('cubs', 'reds', 'yankees', 'dodgers');
print "@teams\n";
But that's not usually the case. More often, you want each element printed on a separate line. To achieve this, you can use this code:
@teams = ('cubs', 'reds', 'yankees', 'dodgers');
foreach (@teams) {

```
print "$_\n";
}
```

Perl uses single or double quotes to surround a zero or more characters. Are the single(' ') or double quotes (" ") identical?

They are not identical. There are several differences between using single quotes and double quotes for strings.

1. The double-quoted string will perform variable interpolation on its contents. That is, any variable references inside the quotes will be replaced by the actual values.
2. The single-quoted string will print just like it is. It doesn't care the dollar signs.
3. The double-quoted string can contain the escape characters like newline, tab, carraige return, etc.
4. The single-quoted string can contain the escape sequences, like single quote, backward slash, etc.

How many ways can we express string in Perl?

Many. For example 'this is a string' can be expressed in:

```
"this is a string"
qq/this is a string like double-quoted string/
qq^this is a string like double-quoted string^
q/this is a string/
q&this is a string&
q(this is a string)
```

How do you give functions private variables that retain their values between calls?

Create a scope surrounding that sub that contains lexicals.
Only lexical variables are truly private, and they will persist even when their block exits if something still cares about them. Thus:

```
{ my $i = 0; sub next_i { $i++ } sub last_i { --$i } }
```

creates two functions that share a private variable. The $i variable will not be deallocated when its block goes away because next_i and last_i need to be able to access it.