

# First!

- I am not Shlomo!
- Who am I (I am the TA)
- Shlomo's excuse: Giving a talk in California, will be back on Wednesday!

# It summer Session!

- Welcome (Greeting from Prof Shlomo)
- Ask yourself, is it better to spend the summer outside or inside on this stuff ?!
- · Hope to be very informal
- I hope to convince you this is more fun

### Today

- Basics (of the course)
  - Overview of the course and objectives
  - Administrative issues

### Basic Perl

- Absolute minimum
- Syntactic details
- Setting up environment (during the break?)

4

# Today

- More Perl
  - Subroutine
  - File I/O
  - Regular Expression
  - Debugging
- Basic Shell Programming (if we have time)

5

6

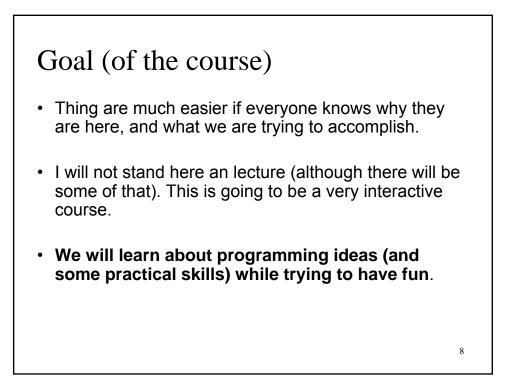
Basic - overview of the class:

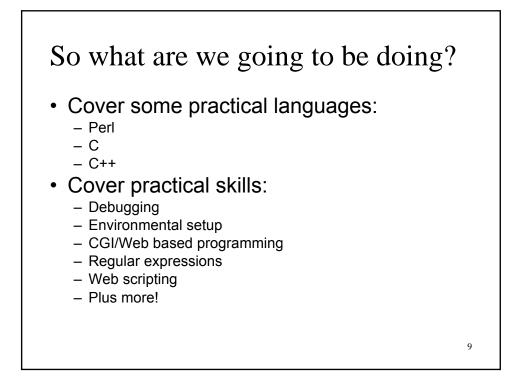
# What?

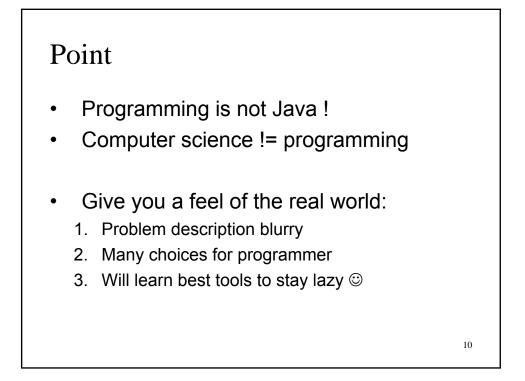
- CS3157: Third course for CS majors.
- Prerequisites:
  - Intermediate knowledge in Programming
  - Object Oriented Programming:
    - What, why, how, and when.
  - Program Designs.
    - Not enough to know how to write the program, need to know how to do it correctly.

7

· Need to learn tool of the trade



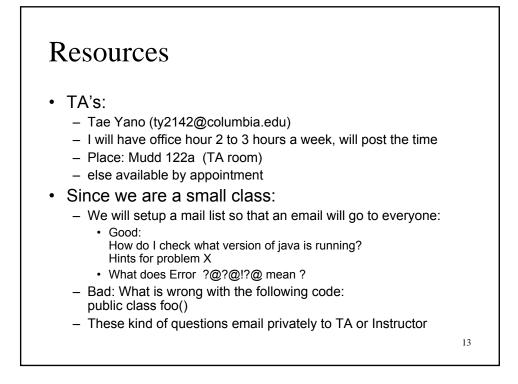


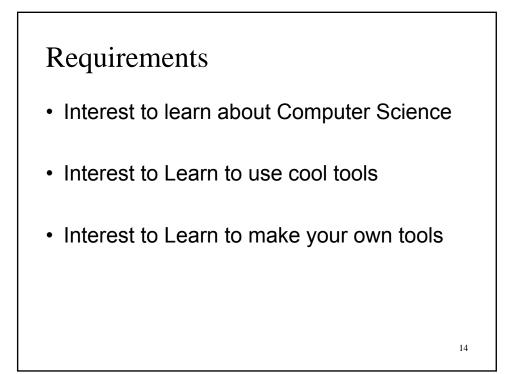


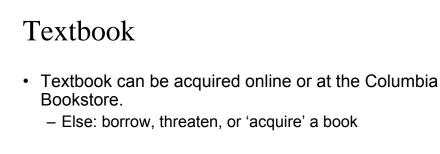
# Basic – admin Issues

# Basics

- Instructor: Professor Shlomo Hershkop (shlomo@cs.columbia.edu)
- Class website:
  - cs.columbia.edu/~sh553/teaching/su06-3157/
  - Check it regularly (at least twice a week).
    - See announcement sections for update info.
  - Will give you my background on Wednesday when we meet
- Meet twice a week: 627 Mudd



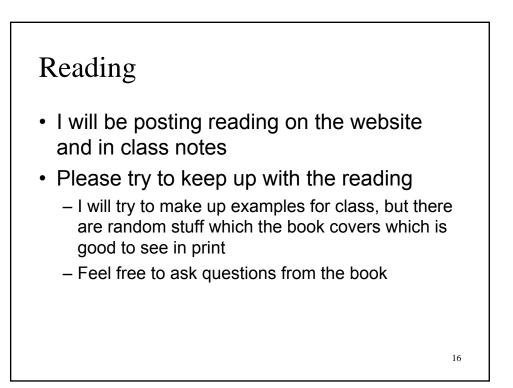




- Perl
  - Programming Perl (get the last edition)
  - Wall, Christiansen, and Orwant
  - O'Reilly

• C

- C how to program
- Dietel and Dietel

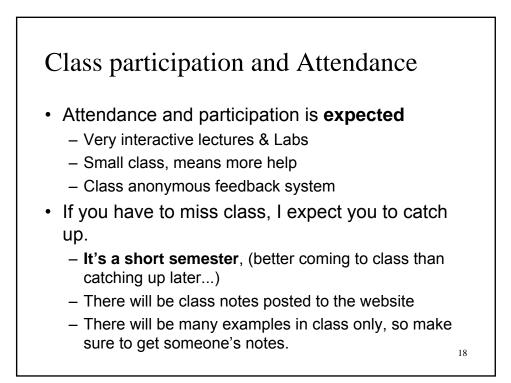


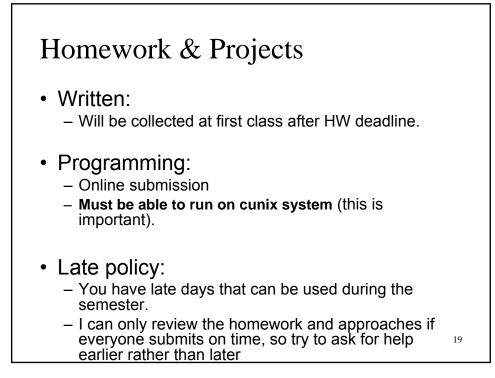


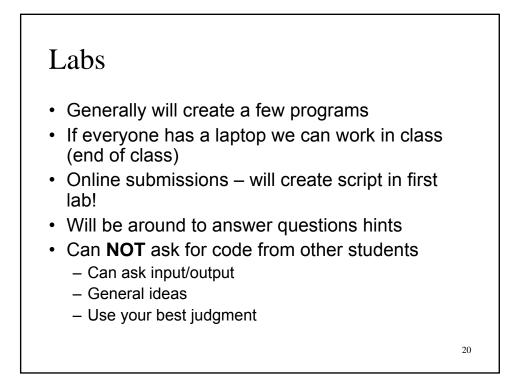
- 6 Labs 120 points
   Out Wednesday, Due by class time
- 2 Homeworks 60 points
   Will have about 2 weeks per homework
- Final (60 points)
   open book
- Homework/Lab is very important:
  - Firm believer in hands on learning
  - Start early
  - Come to office hours, and ask questions

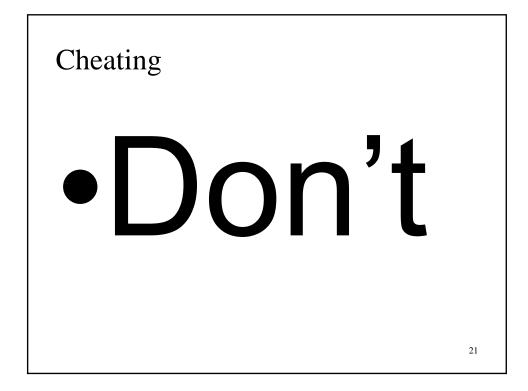
17

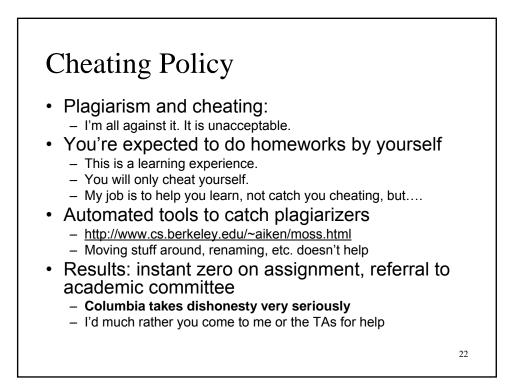
• We are here for YOU!











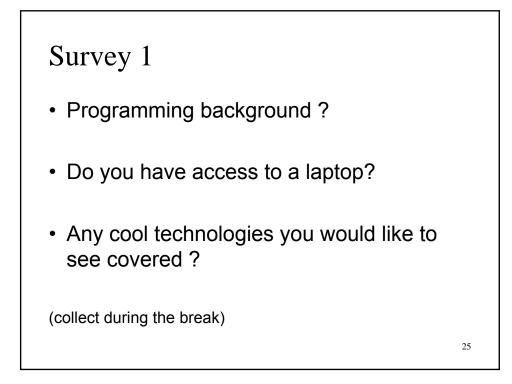
### Feedback System

- Last minute of class will be set aside for feedback:
  - Please bring some sort of scrap paper to class to provide feedback.
  - Feel free to leave it anonymous.
  - Content: Questions, comments, ideas, random thoughts.
- I will address any relevant comments at the beginning of each class
- · Summer is short, so provide feedback !
- Please feel free to show up to office hours or make an appointment at any time

23

# Shopping List

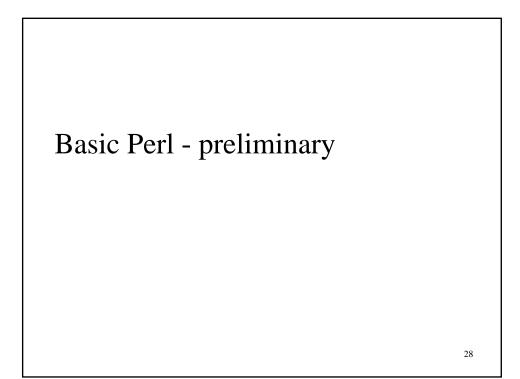
- · You need either a cunix or CS account
  - CS: https://www.cs.columbia.edu/~crf/accounts
  - Try to log into the account asap
- Make textbook plans
  - Recommend : Programming Perl
  - You can choose any perl reference
- Check out the class page



Any Questions ?

# Last plug

- One of the points of computer science is to teach you how to think, learn, and analyze computational related information.
- Example:
  - Task: Create a program to run a web based game, which will be marketed to both desktop and phone users.
  - Any ideas on how to design the programming backend?
  - Ideas on how to measure requirements.
  - What else is important?

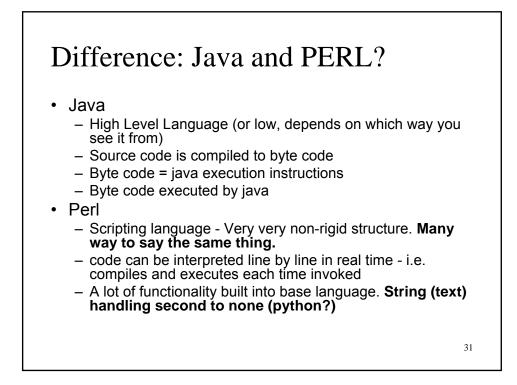


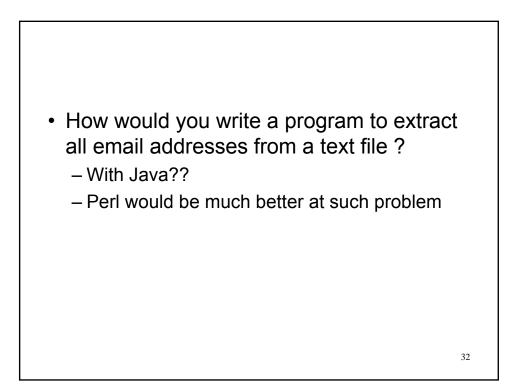
# Perl

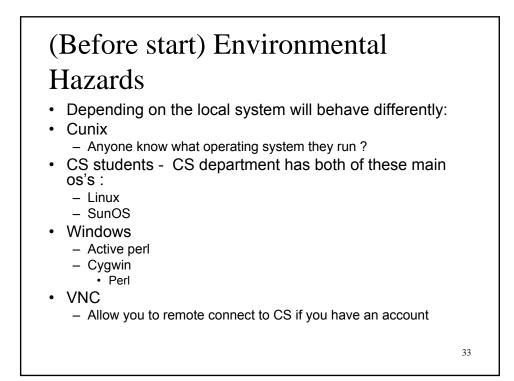
- What is it?
  - Perl was originally designed as a logging tool, released by Larry Wall in 1987.
  - Open source and cross platform. Current version 5.8.7.
  - Referred to as "duct-tape" of the internet
  - Will quickly learn why ©

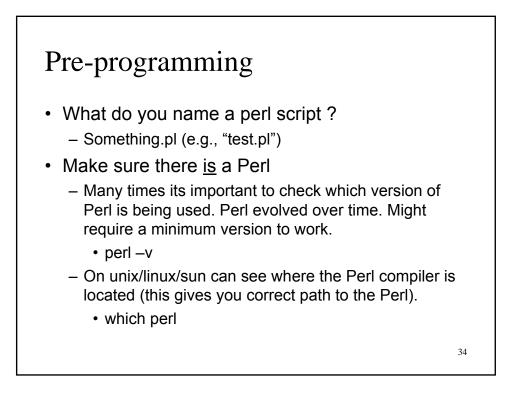
29

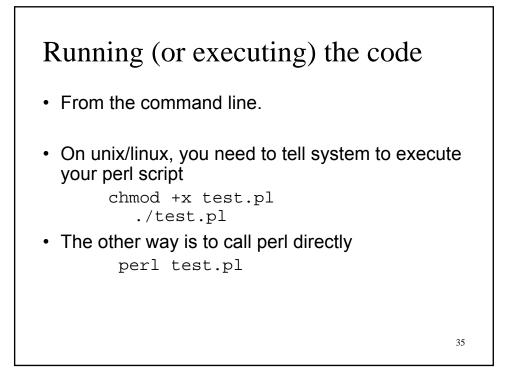
# <section-header><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item>







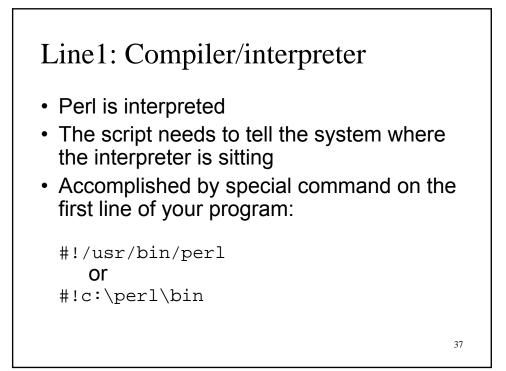


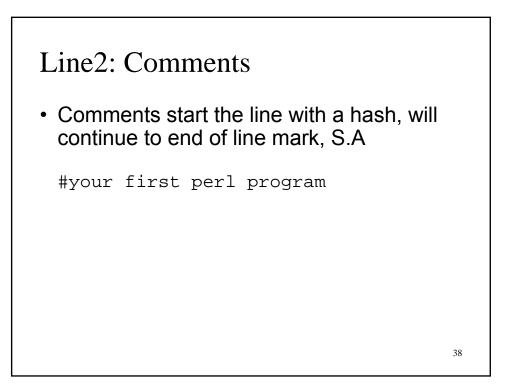


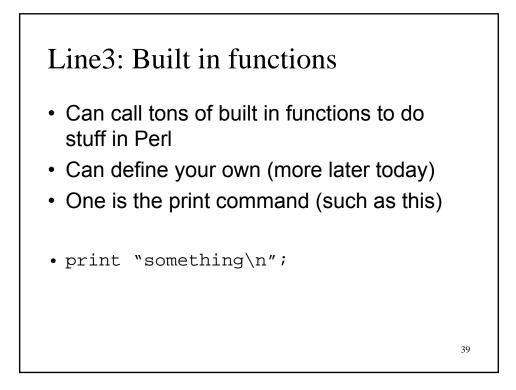
### test.pl

```
#!/usr/bin/perl
#test.pl - should be called
#hello_world.pl
```

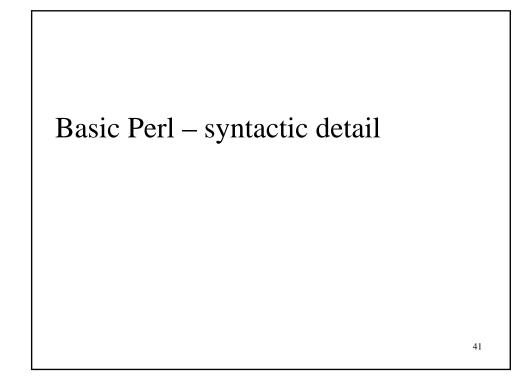
#your first perl program
print "hello everyone\n";

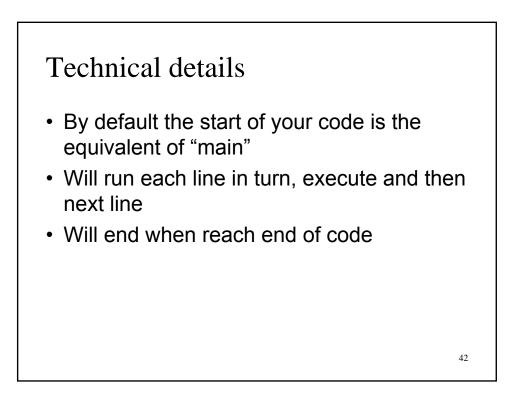


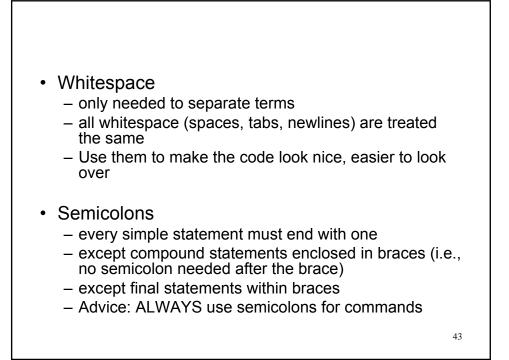


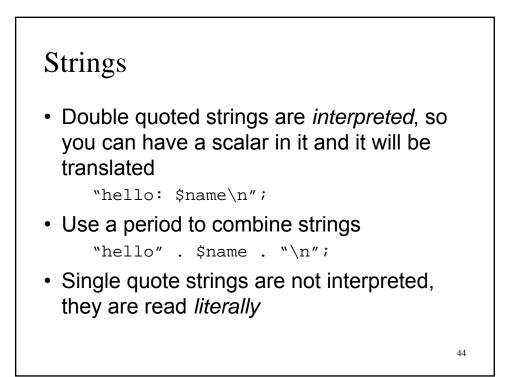


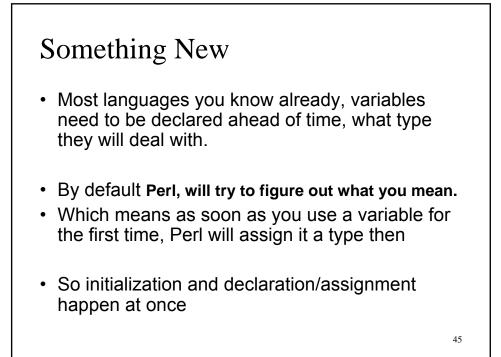
(end of Basic Perl - preliminary) (next – syntactic detail)

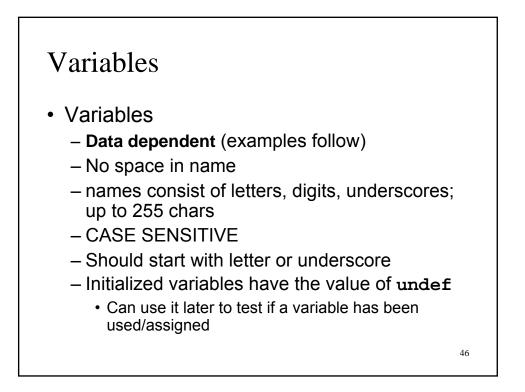


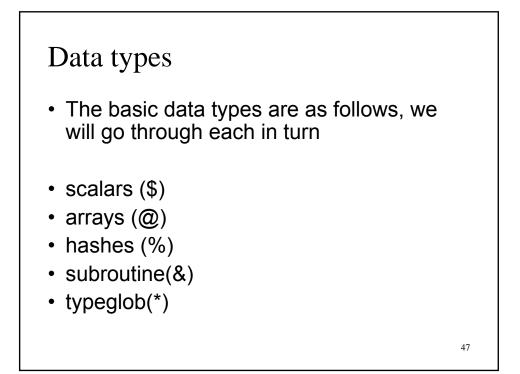








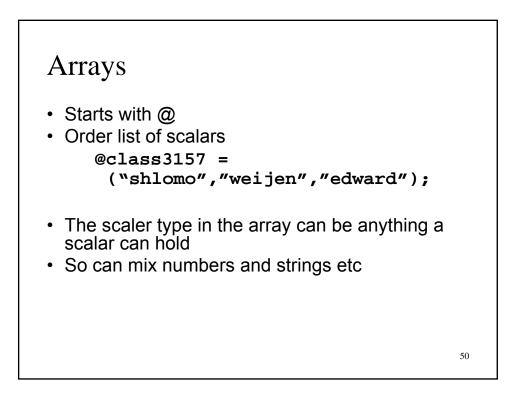


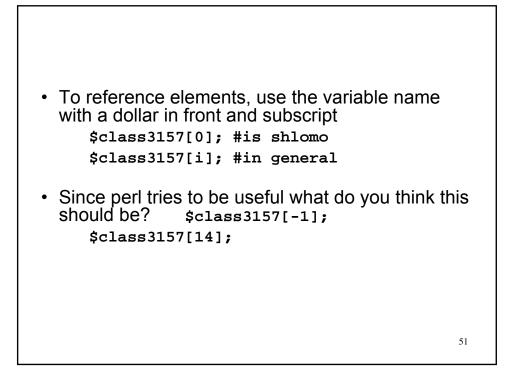


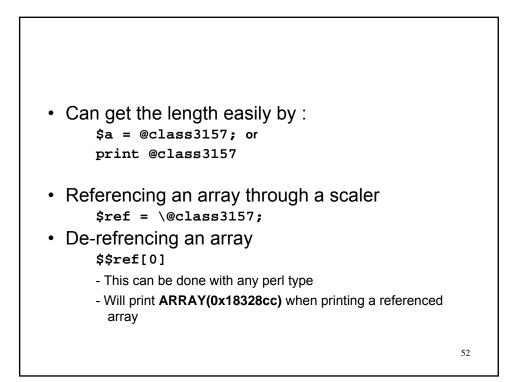
# Scalars

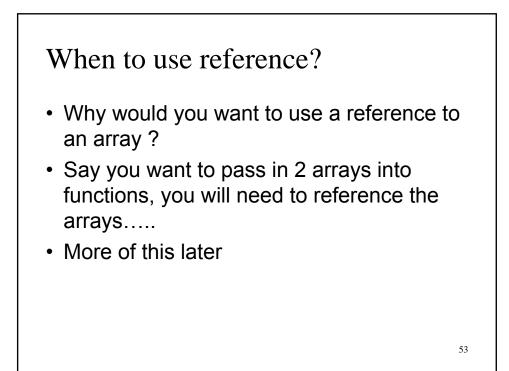
- This type of variable starts with a '\$'
  - \$first
  - \$course
- Can hold: int, real, string
  - 234
  - -89
  - 36.34
  - "hello world"

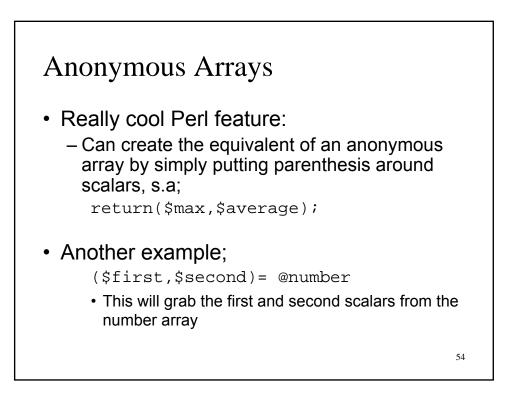
```
s. the set of the
```

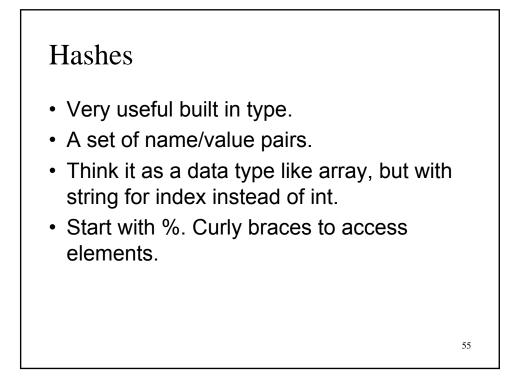


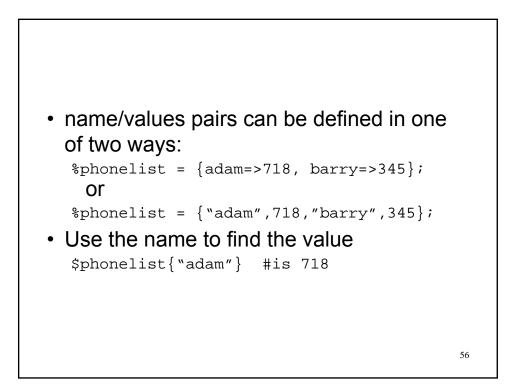


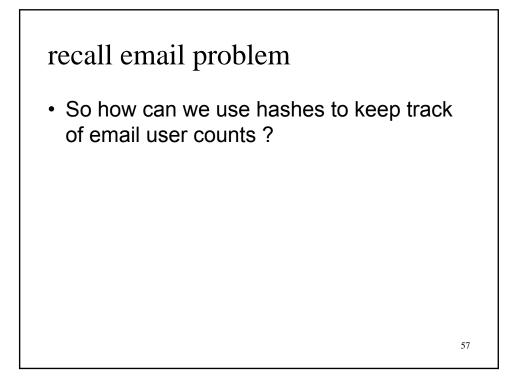




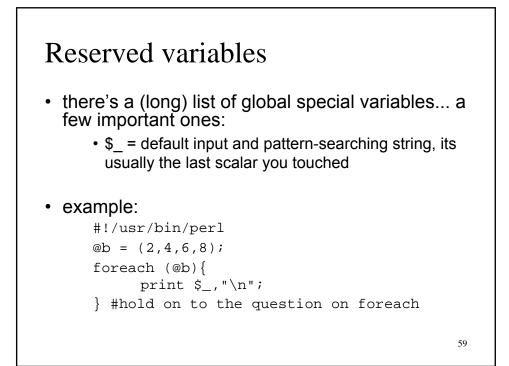


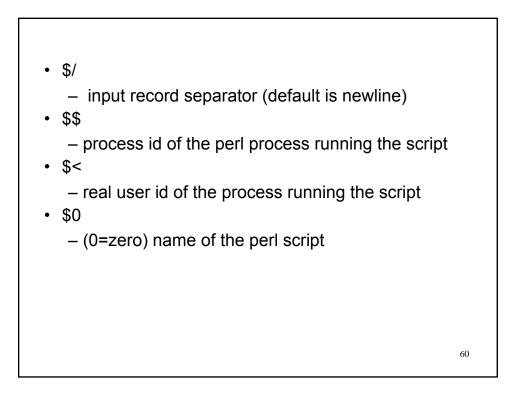


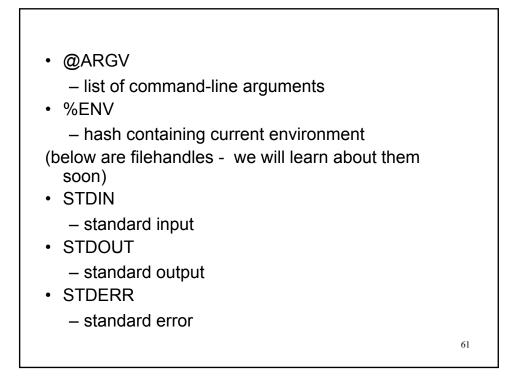


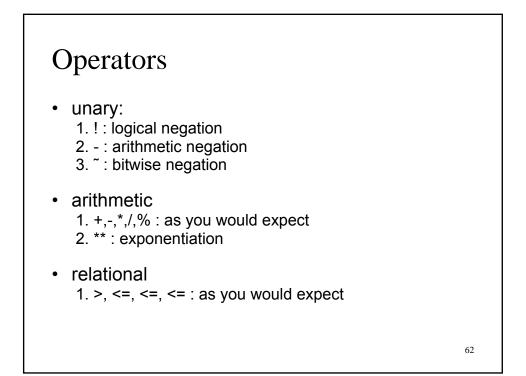


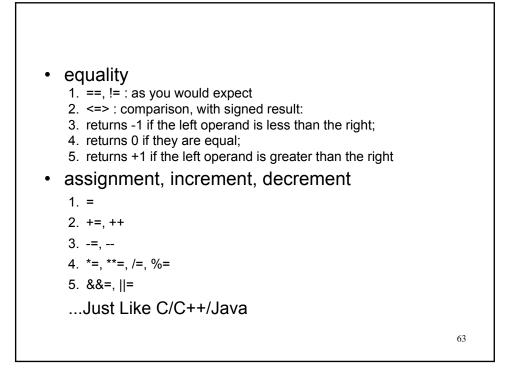
# test2.pl

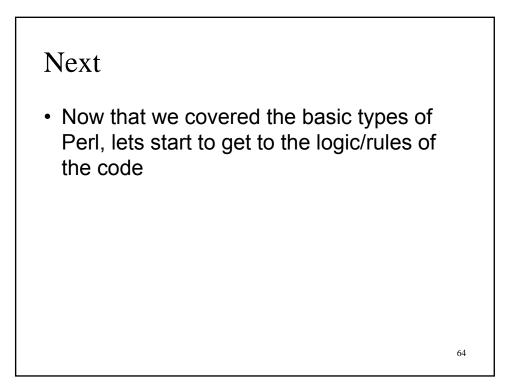


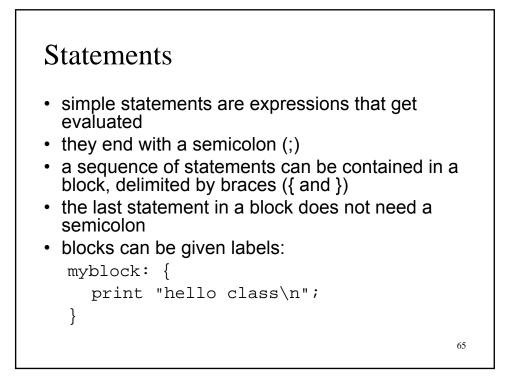


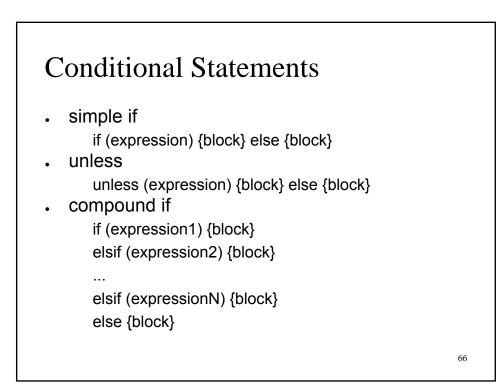


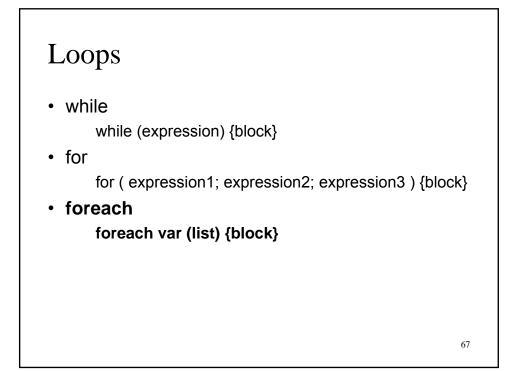










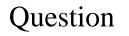


# while

```
Syntax:
   while (expression) {block}
Example:
    #!/usr/bin/perl
   @b = (2,4,6,8);
   $a = @b;
   $i=0;
   while ( $i < $a ) {
      print "i=",$i," b[i]=",$b[$i],"\n";
      $i++;
   }
}</pre>
```

### for Syntax: for (expression1; expression2; expression3) {block} Example: #!/usr/bin/perl @b = (2, 4, 6, 8);\$a = @b; for ( \$i=0; \$i<\$a; \$i++ ) {</pre> print "i=",\$i," b[i]=",\$b[\$i],"\n"; } 69

### foreach The 'foreach' statement allows you to quickly cycle through array values Syntax: foreach var (list) {block} Example: #!/usr/bin/perl @b = (2, 4, 6, 8);\$a = @b; foreach \$e (@b) { print "e=",\$e,"\n"; }



- So if foreach allows you to cycle through arrays...
- How would you cycle through hash, since its composed of key->value pairs ?

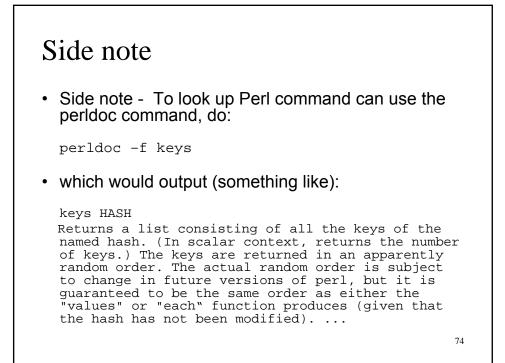
71

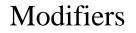
# test3.pl

```
#!/usr/bin/perl
# this is a snip of test3.pl
foreach $k (keys $emailcount) {
   print "$k = "
   print $emailcount{$k};
   print "\n";
}
```

#### keys

- Built in command (or function)
- Allows you to fetch all the keys of the hash type
- · Use each one to access the individual value pair





- Nifty grammar, but should be careful about assuming a line will execute, read it carefully. (There are many way to say same thing...)
- You can follow a simple statement by an if, unless, while, or until modifier:

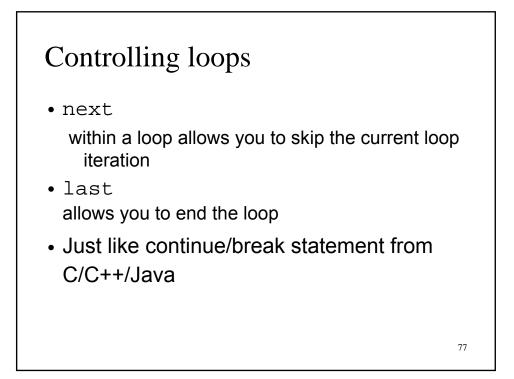
statement *if* expression; statement *unless* expression; statement *while* expression; statement *until* expression;

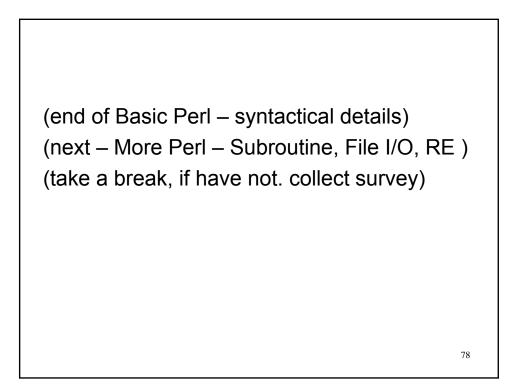
75

#### Examples

• Examples:

```
#!/usr/bin/perl
@b = (2,4,6,8);
$a = @b;
print "hello world!\n" if ($a < 10);
print "hello world!\n" unless ($a < 10);
#print "hello world!\n" while ($a < 10);
#print "hello world!\n" until ($a < 10);</pre>
```





### More Perl – Subroutine, File I/O, RE

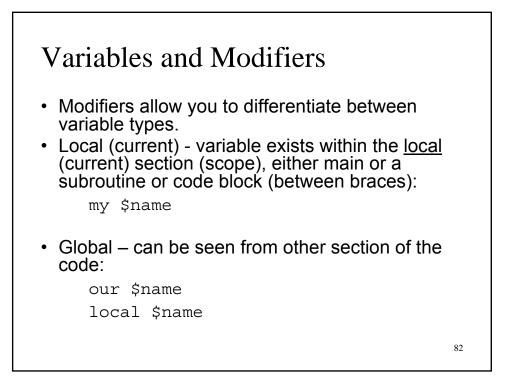
#### Ok so far

- We have variable types, arithmetic operators, and some logic
- More interesting parts. Starting with writing your own functions, which Perl calls *subroutines*
- Talk of scope before going into subroutine...

80

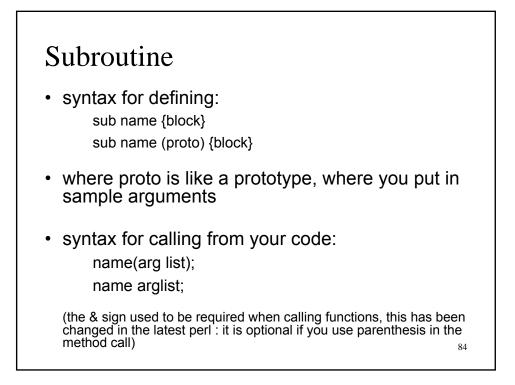
#### Scope

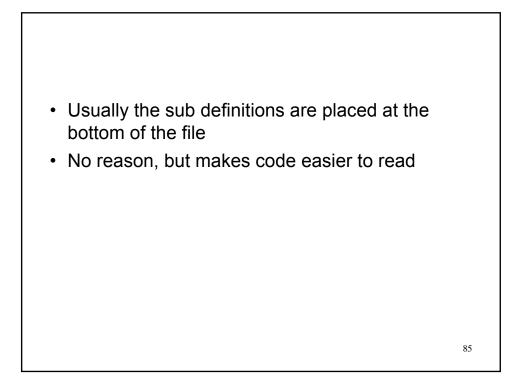
- What is scope?
- Default scope is main
- \$name can also be referred to as \$main::name
- package NAMESPACE
   Within any block of code, can declare that the rest of the code will belong to a specific namespace



#### Scope: usage

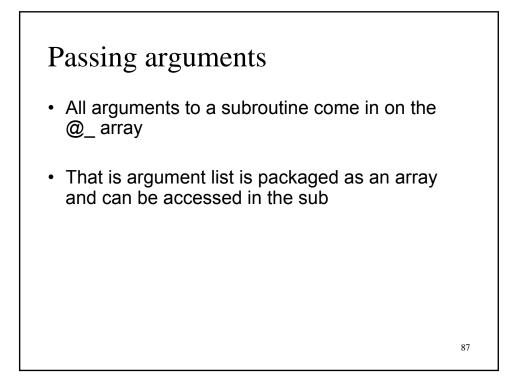
- my \$time, \$out;
  here, only time is a local variables
- my (\$time,\$out)
  - correct way to do it.
- Remember to place more than one variable in parenthesis!!

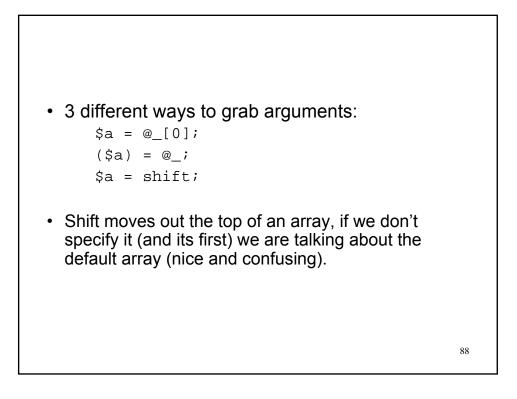


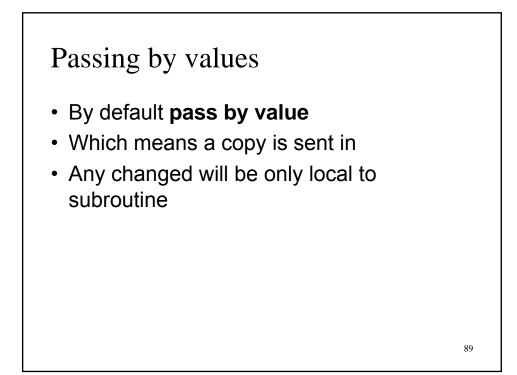


#### sub1.pl

```
print "welcome to the program";
testsub();
sub testsub(){
  print "hi everyone\n";
}
```

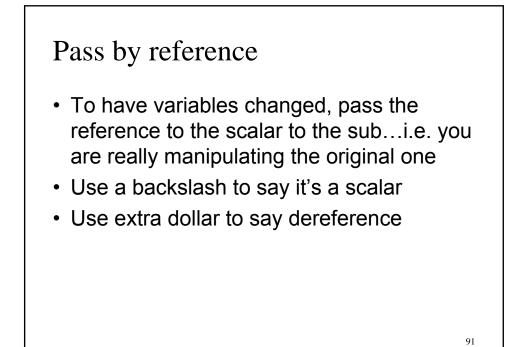






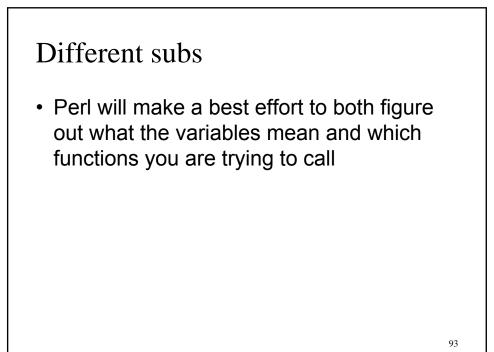
#### So...what will be printed here?

```
$n = 45;
print "n is now $n\n";
testsub($n);
print "n is now $n\n";
sub testsub{
    $a = shift;
    print "in testsub 1 = $a\n";
    $a++;
    print "in testsub 2 = $a\n";
}
```



#### Pass by reference

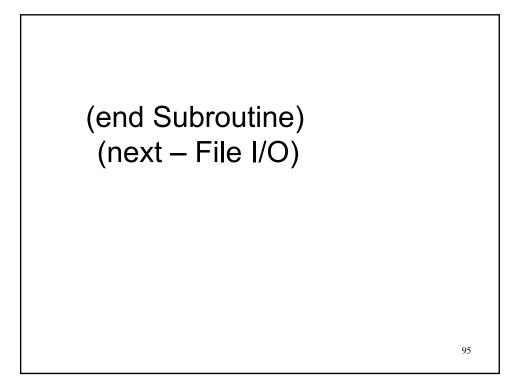
```
$n = 45;
print "n is now $n\n";
testsub(\$n);
print "n is now $n\n";
sub testsub{
    $a = shift;
    print "in testsub $a\n";
    $$a++;
}
```

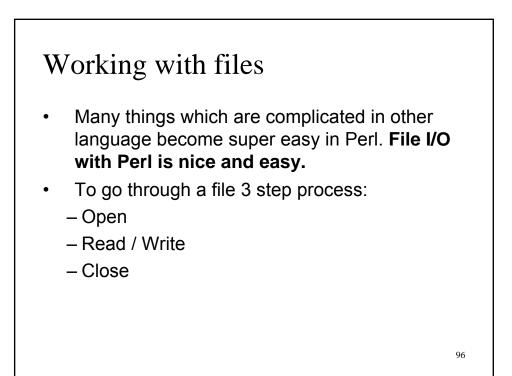


#### Example

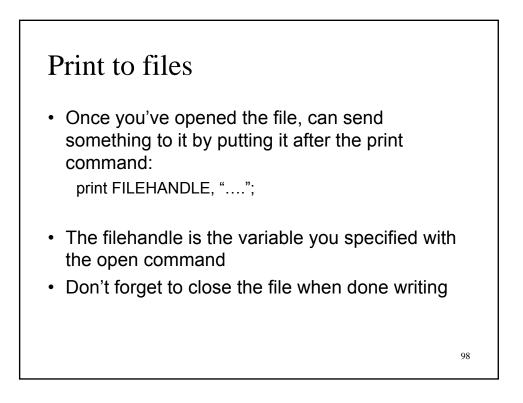
```
#!c:\perl\bin
#this is sub4.pl snip
($first,$last) = &getname();
print "First is $first";
#return the fill name as a string
sub getname(){
   return "shlomo hershkop";
}
#return name split
sub getname(){
   return ("shlomo","hershkop");
}
```

47





Open files
<ul> <li>When you open, you need to say what type of operations you will be doing:</li> </ul>
open( FILEHANDLE, filename ); # to open a file for read in open( FILEHANDLE, >filename ); # to open a file for writing open( FILEHANDLE, >>filename ); # to open a file for appending
97



#### Example:

```
#!/usr/bin/perl
#this is the snip from fh1.pl
open( MYFILE,">a.dat" );
#for example of s.c, see the next
print MYFILE "hi there!\n";
print MYFILE "bye-bye\n";
close( MYFILE );
```

```
<section-header><list-item><text><text><text><list-item><list-item><list-item>
```



- · So what if the open command fails?
- A trick use double pipe to do something like this;

101

... || warn print "message";

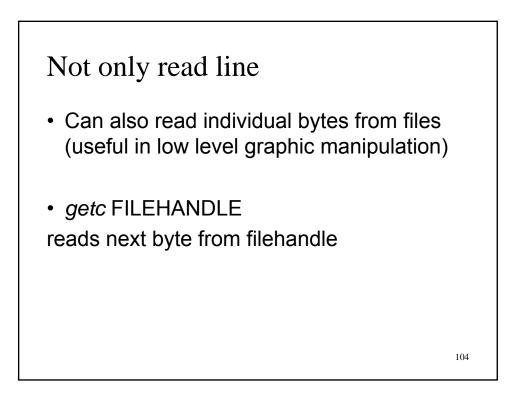
• Or if you want to fail: ... || die print "message";

Example II

```
#!/usr/bin/perl
open( MYFILE1, "a.dat" ) || warn "file 1
not found!";
open( MYFILE2, ">b.dat" ) || die "file 2
problems!";
while ( <MYFILE1> ) {
    print MYFILE2 "$_\n"
    }
    close( MYFILE2 );
```

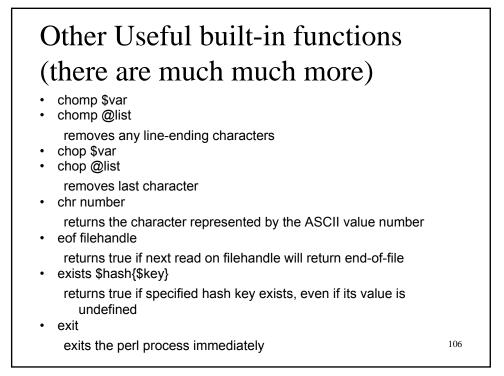
#### Take a second to look this over

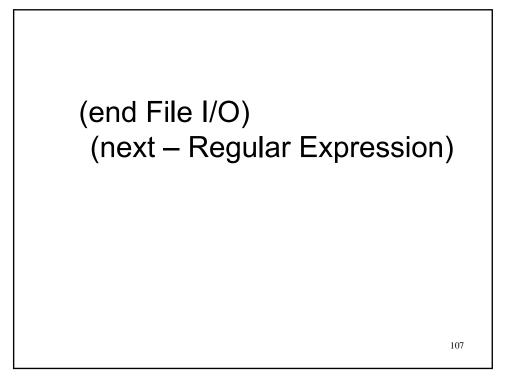
```
#!/usr/bin/perl
open( TEST, "test.txt" ) || die "can not
  open file!\n";
$linecount =0;
while ($line = <TEST>){
    $linecount++;
}
close( MYFILE );
print "number of lines in the file:
    $linecount\n";
```

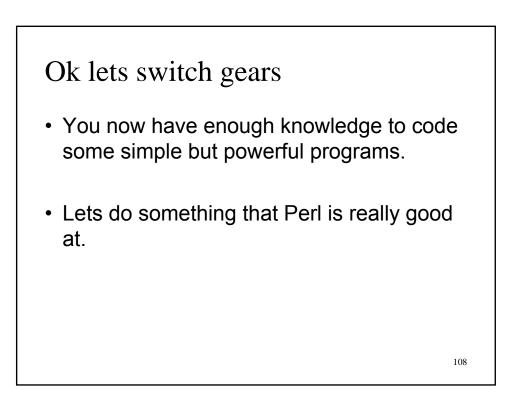


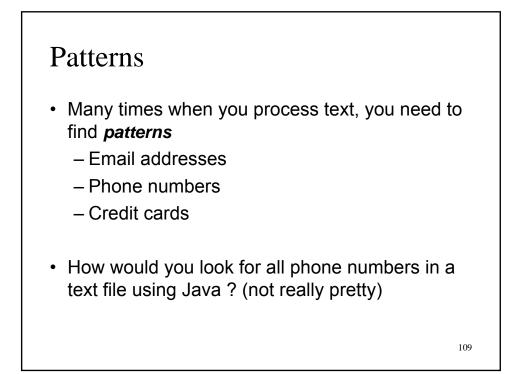
#### chomp

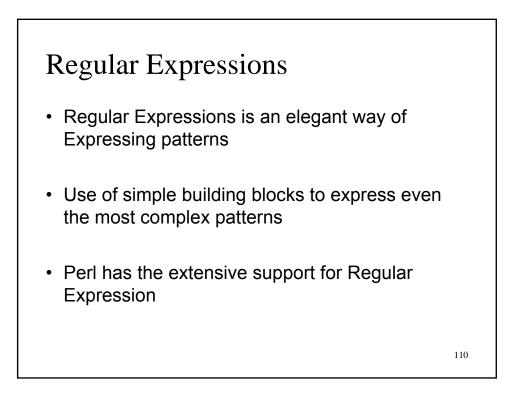
- Not lunch! Another built-in function
- Removes \n very useful when processing logs
- V. useful when you are reading line from file

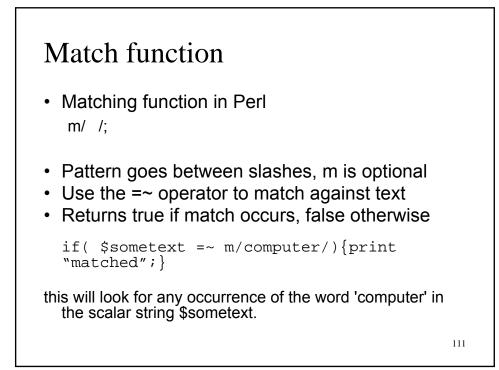


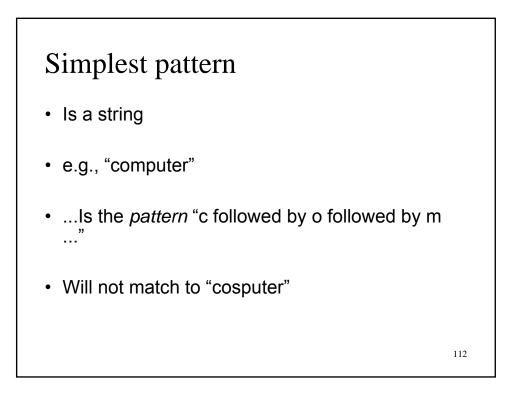












#### Example (simple)

```
What will this do ?
#!c:\perl\bin
# this is snip of rel.pl
$name = "shlomo hershkop";
if($name =~ /lom/){
   print "have found match\n";
}
else{
   print "no match found\n";
}
```

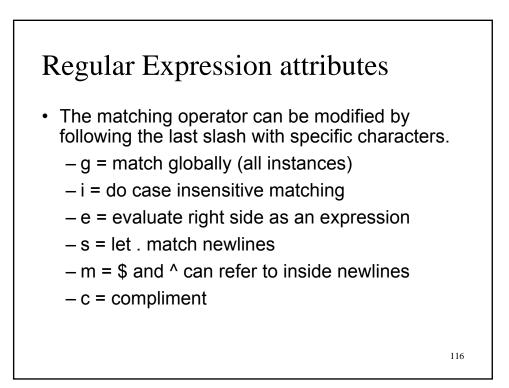
113

#### Example 2

```
#!/usr/bin/perl
$s = "hello world";
print '$s=[',$s,"]\n";
$t = ($s =~ s/l/x/g);
print '$t=[',$t,"]\n";
print '$s=[',$s,"]\n";
# this outputs:
# $s=[hello world]
# $t=[3]
# $s=[hexxo worxd]
```

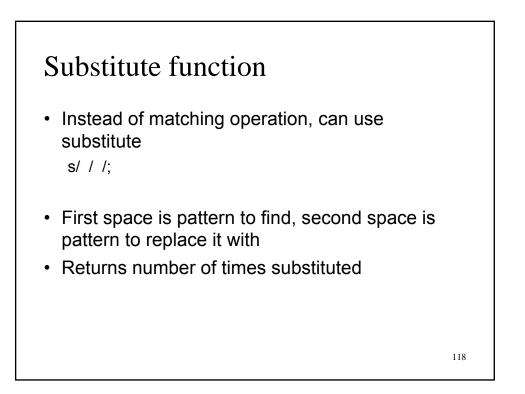


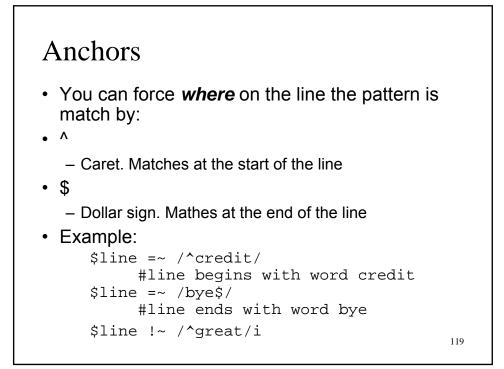
- Can flip the match by saying:
   !~/ /
- Example: \$line !~ /great/
- Will match any line with the word 'great'

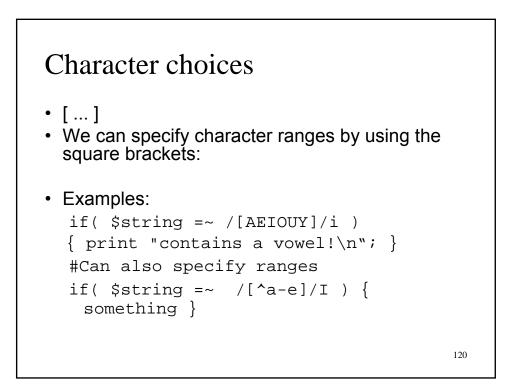


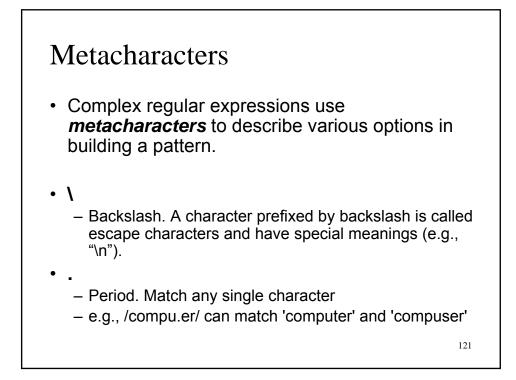
#### Example 1

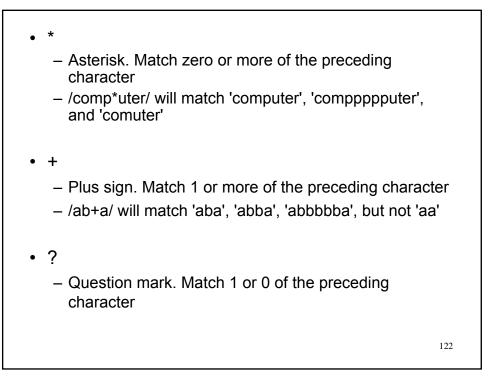
```
#!/usr/bin/perl
$s = "hello world";
print '$s=[',$s,"]\n";
if ($s =~ m/x/)
  { print "there's an x in ",$s,"\n" }
else
  { print "there isn't\n" }
if ($s =~ m/L/i)
  { print "there's an l in ",$s,"\n" }
else
  { print "there isn't\n" }
# this outputs:
# $s=[hello world]
# there isn't
# there's an l in hello world
```











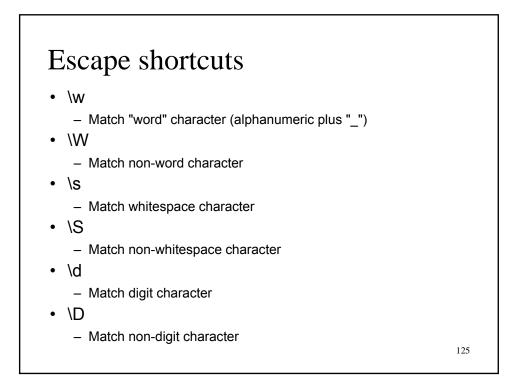
#### Quantifiers

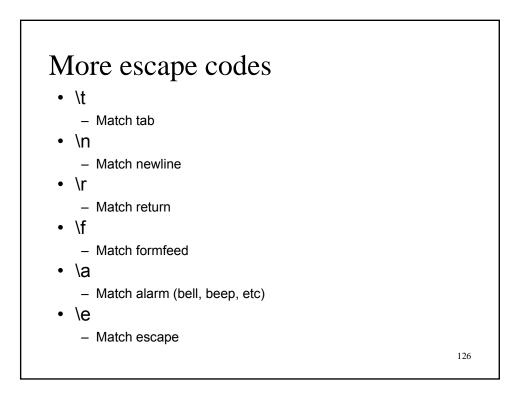
- { n1, n2 }
  Match n1 to n2 of the preceding character
- { n, }
   Match n or more of the preceding character
- { n }
   Match exactly n of the preceding character

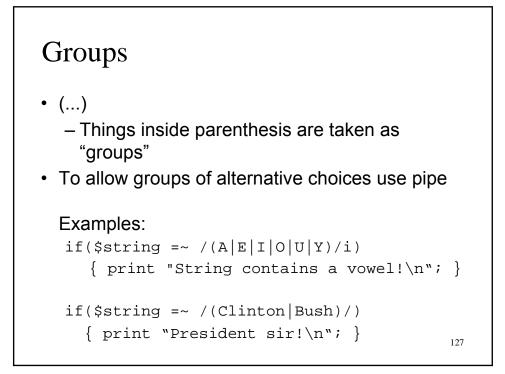
123

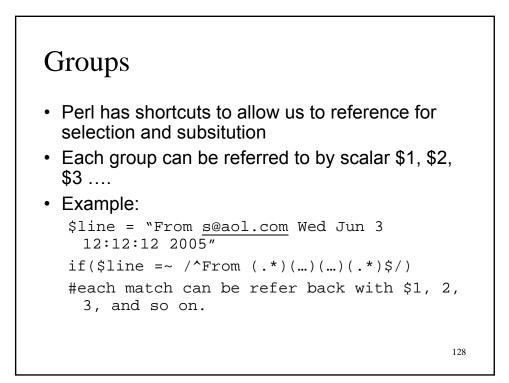
#### Examples:

/ba*b/;	#b,zero or more a, b
/ba{3,5}b/;	#b, 3 and 5 a's, b
/ba{2}b/;	<pre>#b, exactly 2 a's, b</pre>
/(ab){4,}/;	#4 or more ab's
/[a-h]{1,4}/;	#1 to 4 character a~h

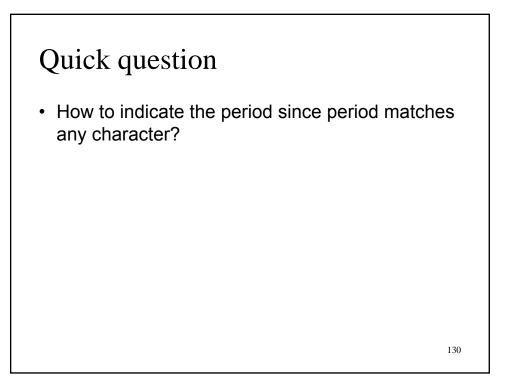








# <section-header><text>



#### Usage Example

```
if ( $line =~ /^\s.*\S$/ ) {...}
if ( not $line =~ /cs3157/ ) {...}
if( $line !~ /cs3157/ ) {...}
while ( $line =~ /^\w \w$/) {...}
```

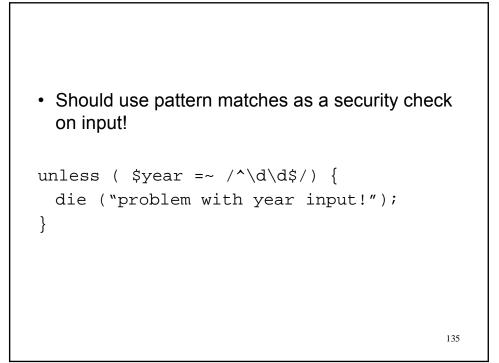
131

#### What is?

```
open MAIL, "mail.txt" or die "cant open
file\n";
while(<MAIL>) {
  print if m/^From: /;
}
```

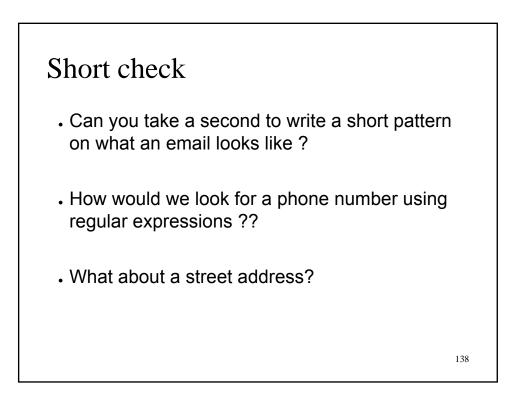
```
open MAIL, "Mail.txt" or die "can't open
mail file\n";
while (<MAIL>) {
    if (/^([^:]+): ?(.+)$/ ) {
        print "Header $1 has val $2\n";
    }
```

```
if($string =~
 m/^\S+\s+(Hershkop|Stolfo|Aho)/i){
 print "$string\n"
 };
```

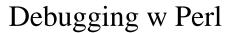


#### What is? prame = "advanced programming class" if(\$name =~ /programming) { print \$ ` ; print \$ & ; print \$ ' ; }

```
<section-header><section-header><text><text><code-block><text></code>
```

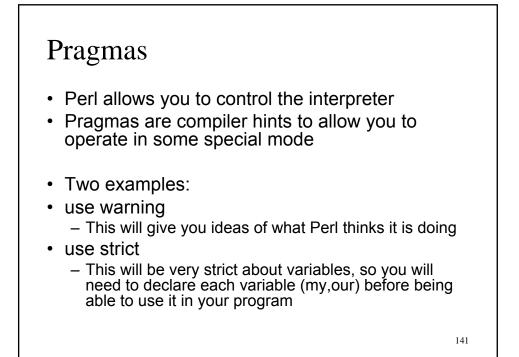


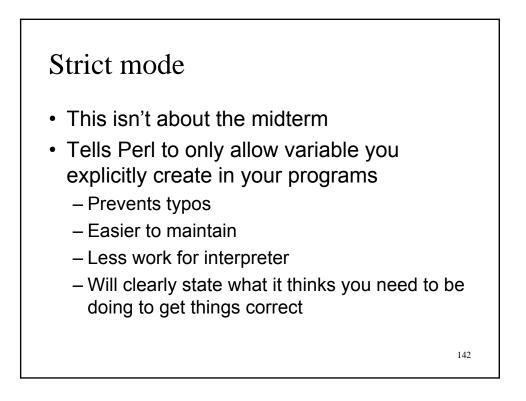
## (end of RE practice) (next - debugging)

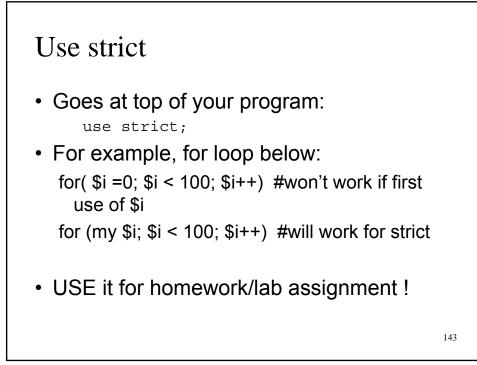


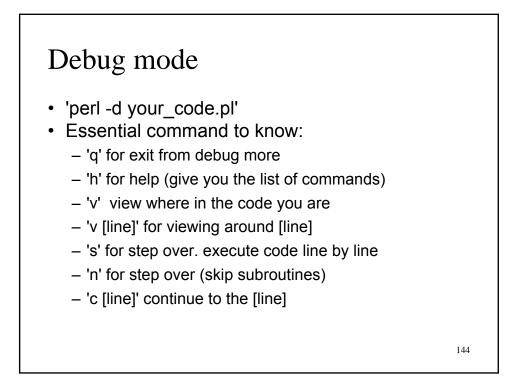
- Use 'strict' pragma
- Use 'warning' pragma
- Use 'perl -w' option
  This output warning as it compile
- Use 'perl -c' option
  - This does syntax check
- Use debugger '-d' option
  - bring you down (up?) to debug mode.

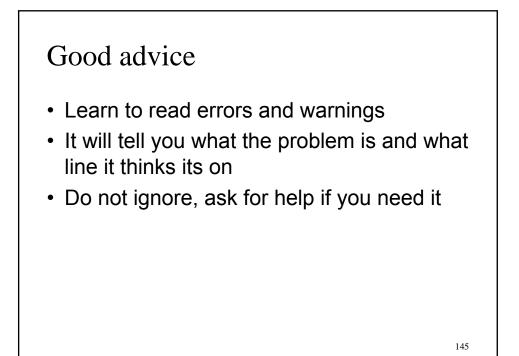
140

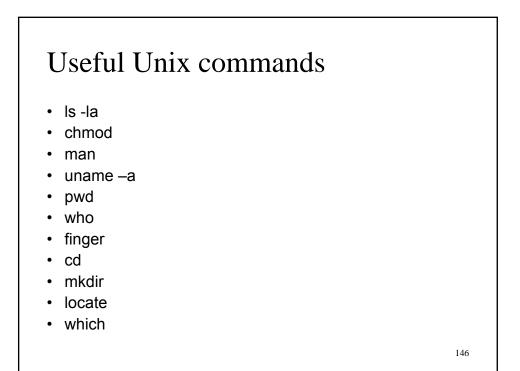














 there are lots and lots of advanced and funky things you can do in perl; this is just a start!

here's a quick start reference:

- http://www.comp.leeds.ac.uk/Perl/
- http://www.perl.com

function reference list is here:

http://www.perldoc.com/perl5.6/pod/perlfunc.htm

147

#### Perldoc

- Use 'perldoc', or 'man'
- Below are some of the helpfulones:
  - perlre (1) about Regular Expression
  - perlvar (1) predefined variables explained here
  - perlrun (1) about command line option
  - perlop (1) about operators
  - perlfunc (1) about built in functions
  - perldebtut (1) debugger help