

topics:

CGI and perl

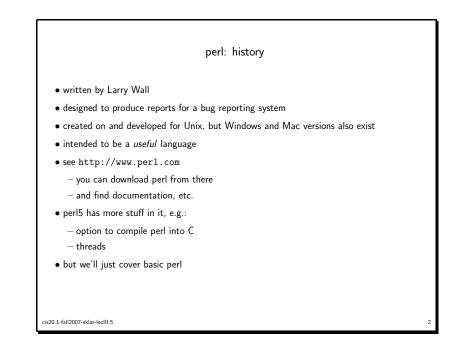
cis20.1-fall2007-sklar-lecIII.5

perl: basics

• first line of file is

#!/usr/bin/perl

- this is the path to the perl executable
- if it doesn't work, then do which perl to find out where perl is installed on your system
- the perl executable runs the perl interpreter, to interpret and execute your perl script
- the interpreter converts script to bytecode prior to execution, so it is sort of like a compiler (although bytecode is not stored anywhere)
- make the script executable (chmod +x <filename>), like your shell scripts from last week

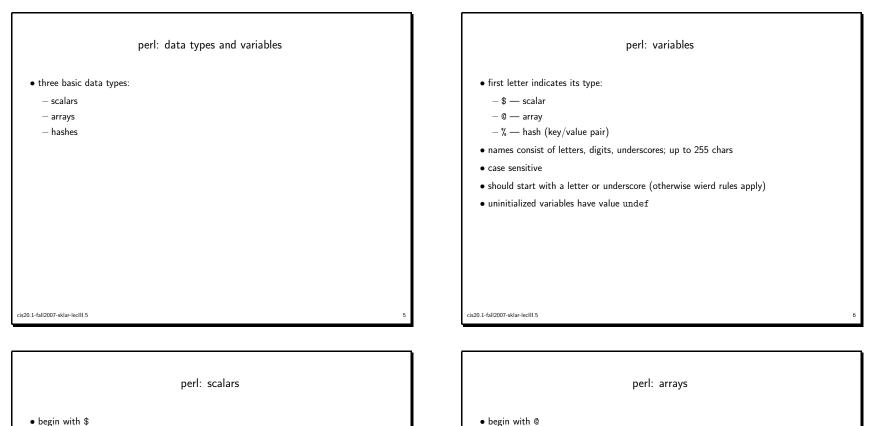


perl: program structure

- whitespace
 - only needed to separate terms
 - all whitespace (spaces, tabs, newlines) is the same
- semicolons
 - every simple statement must end with one
 - $-\ensuremath{\mathsf{except}}$ compound statements enclosed in braces (i.e., no semicolon needed after the brace)
 - except final statements within braces
- declarations
 - only subroutines and report formats need explicit declarations
 - otherwise, variables in perl are like in shell scripts they are declared and initialized all at once
- comments
 - from hash (#) to end of line

cis20.1-fall2007-sklar-lecIII.5

cis20.1-fall2007-sklar-lecIII.5



- numbers
 - integers
 - floating point
 - -e.g., 123, -456, 0xff, 3.14, 4_567
- strings
 - delimited by single or double quotes
 - -e.g, "123", "abc", 'alphabet'

cis20.1-fall2007-sklar-lecIII.5

• ordered list of scalar values

• e.g.: \$fruit[0] is "apple"

-1 is the last element in the list

square brackets

• e.g.: @fruit = ("apple", "orange", "pear");

• negative subscripts count backwards from the last element;

• refer to single element using \$ in front of name (in place of @) and index of element in

cis20.1-fall2007-sklar-lecIII.5



(\$sec,\$min,\$hr,\$mday,\$mon,\$yr,\$wday,\$yday,\$isdst) = localtime(); print "s=",\$sec," min=",\$min," hr=",\$hr," mday=",\$mday, " mon=",\$mon," yr=",\$yr," wday=",\$wday, " yday=",\$yday," isdst=",\$isdst,"\n";

```
$today = localtime();
print "today=",$today,"\n";
```

output:

s=31 min=29 hr=21 mday=2 mon=2 yr=103 wday=0 yday=60 isdst=0 today=Sun Mar 2 21:29:31 2003

cis20.1-fall2007-sklar-lecIII.5

```
perl: contexts, example 2

• example

#!/usr/bin/perl

$a = (2,4,6,8);
print '$a=',$a,"\n";

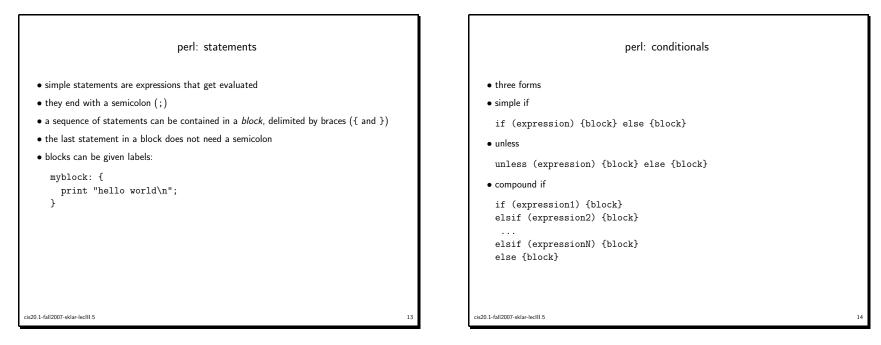
@b = (2,4,6,8);
print '@b=',@b,"\n";

$a = @b;
print '$a=',$a,"\n";

• output

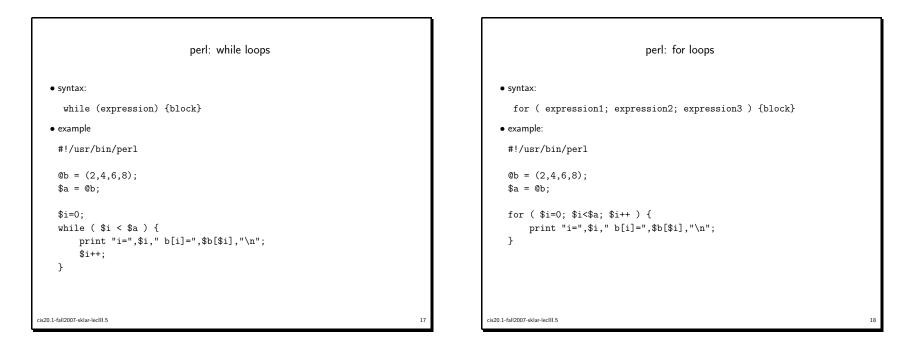
$a=8
@b=2468
$a=4

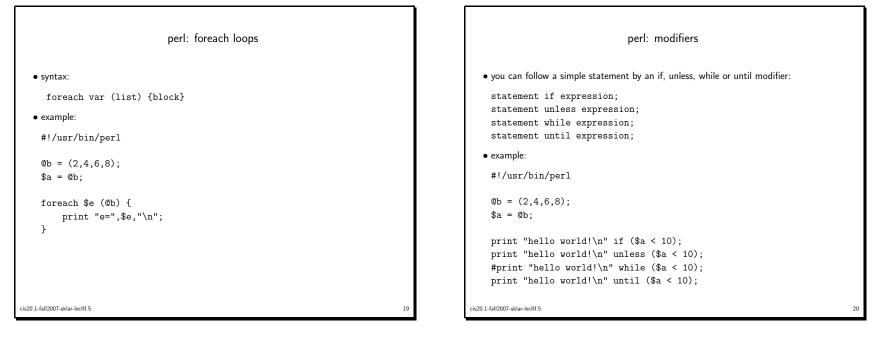
cseo1-fall2007-sklar-teell1.5
```

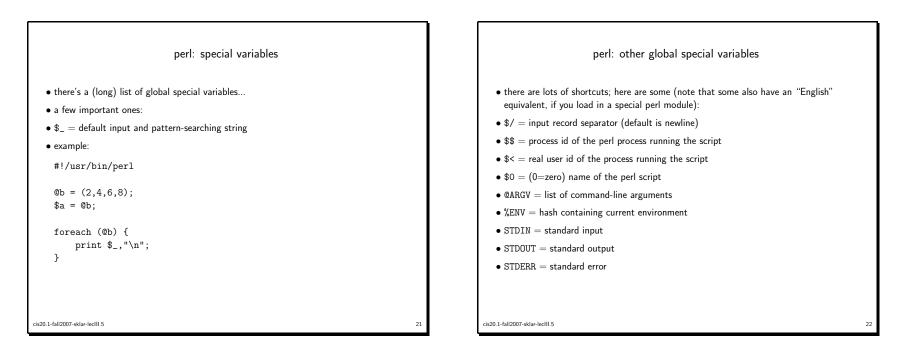


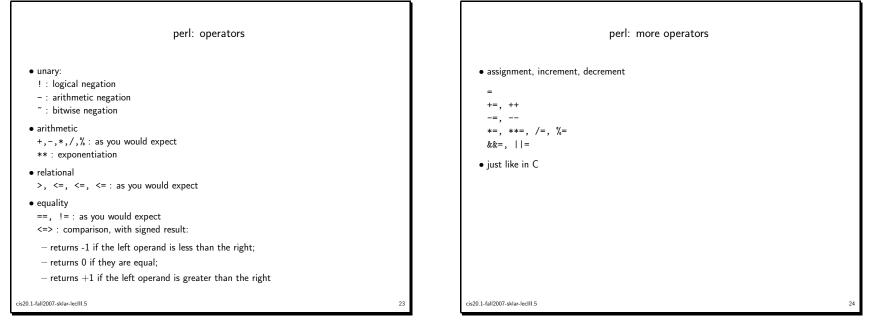
perl: conditionals, example
#!/usr/bin/perl
<pre>@b = (2,4,6,8); \$a = @b;</pre>
if ($a > 0$) { print "a is greater than 0!\n" } else { print "a is NOT greater than 0!\n" }
unless ($a > 0$) { print "a is NOT greater than 0!\n" } else { print "a is greater than 0!\n" }
if ($a > 0$) { print "a is greater than $0!\n"$ } elsif ($a < 0$) { print "a is less than $0!\n"$ } else { print "a is exactly $0!\n"$ }
cis20.1-fall2007-sklar-lecll1.5

	perl: loops	
• while		
forforeach		
cis20.1-fall2007-sklar-lecIII.5		16









perl: files, aka filehandles	
<pre>• open(FILEHANDLE, filename); : to open a file for reading open(FILEHANDLE, >filename); : to open a file for writing open(FILEHANDLE, >>filename); : to open a file for appending • use warn print "message"; or die print "message"; for error checking • print FILEHANDLE,; • close(FILEHANDLE,; • close(FILEHANDLE); • example:</pre>	<pre>#!/usr/bin/ open(MYFII open(MYFII while (<my close(="" myf]<="" pre=""></my></pre>
cis20.1-fall2007-sklar-leclII.5 25	cis20.1-fall2007-sklar-l

		CCI
perl	and	CGL

- depending on how the web server is set up, you may need to name your perl file <filename>.cgi instead of <filename>.pl
- you may also need to put the file in a special directory called cgi-bin which may reside in your public_html directory tree or in the main web server directory tree (typically /var/www/cgi-bin/)
- \bullet the main thing you need to know is how to get values from HTML forms into perl scripts
- \bullet this can be done using either the POST or GET methods
- the GET method puts variable values into the QUERY_STRING environment variable, which can be grabbed in perl using the %ENV has, as follows:

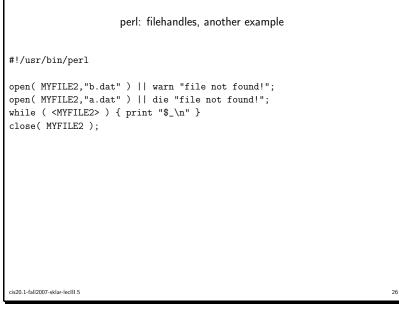
\$input = \$ENV{'QUERY_STRING'}

• the POST method sends variable values from the form to the recieving action script via stdin, which can be read in perl as follows:

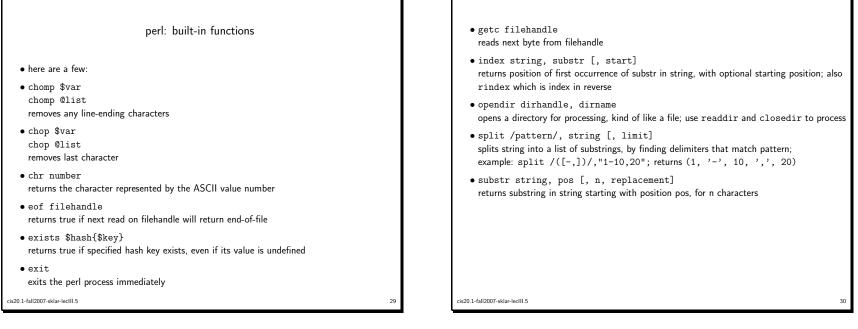
2

\$input = <STDIN>;

cis20.1-fall2007-sklar-lecIII.5



perl: subroutines	
• syntax for defining:	
sub name {block} sub name (proto) {block}	
• where proto is like a prototype, where you put in sample arguments	
• syntax for calling:	
<pre>name(args); name args;</pre>	
• any arguments passed to a subroutine come in as the array @_	
\bullet you can use the return statement, like in C	
ris20.1-fall2007-sklar-leclII.5	28



cis20.1-fall2007-sklar-lecIII.5

 simplest regular expression is a literal string complex regular expressions use <i>metacharacters</i> to describe var pattern "I never metacharacter I didn't like" (1) escapes the character immediately matches any single character exceed matches at the beginning of a string matches the preceding element 0 of a string matches the preceding element 1 of the specifies a range of occurrences for the second string the second s	following it pt newline
 pattern "I never metacharacter I didn't like" escapes the character immediately matches any single character excee matches at the beginning of a string matches at the end of a string matches the preceding element 0 metacharacters: + matches the preceding element 1 matches the preceding element 0 	following it pt newline
 matches any single character exce matches at the beginning of a string matches at the end of a string matches at the end of a string matches the preceding element 0 matches the preceding element 1 matches the preceding element 0 	pt newline
[] matches any one of the class of cl () groups expressions	or more times or more times or 1 times or the element preceding it naracters in the brackets
note that there are some exceptions to these rules	

perl: pattern matching • = " binds a scalar to a patterm match, substitution or translation • ! ~ just like above, except that the return value is negated in the logical sense • operators: - m/pattern/gimosx : match * g = match globally (all instances) *i = do case insensitive matching * note that first m is optional - s/pattern/replacement/egimosx : search * e = evaluate right side as an expression* g = match globally (all instances) *i = do case insensitive matching - y/pattern1/pattern2/cds : translate * c = complement pattern1 * d = delete found but unreplaced characters* s = squash duplicate replaced characters

