2) Scalar data

Matt Webster
IMBIM, BMC
matthew.webster@imbim.uu.se

website

- http://blog.websterlab.eu/courses/perl/

lectures, exercises, course plan etc.

adding comments

- Comments: The # symbol, and anything from it to the end of the line is ignored.

```
# get start and stop values from the user
$start = <STDIN>
$stop = <STDIN>
# calculate string length
$length = $start - $stop + 1;
```

What is a scalar?

- Simplest kind of data in perl
- used for storing a single item of data
- a number, or a string
- perl automatically converts:
  - i.e. `10` can be `'10'`

- `$example`

is a scalar variable

Numbers

```
0
1.2
1e4
-10.32e-34
((23 + 13) / (24 - 12)) * 283

$example = 52 * 2;
```

Strings

- single quoted `'string'`
  - two special characters beginning with backslash
  - `\'` means '
  - `\\` means \

```
'a string written by Matt'
```

```
a string written by Matt
```

```
'Matt\'s string'
```

```
Matt's string
```

Strings

- double quoted `"strings"`
  - many special characters beginning with backslash
  - `\n` means newline
  - `\t` means tab

```
"Hello world\n"
Hello world
"10\t20\t30\n"
10  20  30
```

backslash escape

\

backslash is used to make many different characters within double quotes

Construct	Meaning
\n	Newline
\r	Return
\t	Tab
\\	Backslash
\”	Double quote

early warning!!

perl expects files to have \n at the end of lines

some files have \r (or \r\n)
this causes headaches!

don't use \r !

examples of double-quoted strings

```
"hello world";
  hello world
"hello\tworld";
  hello    world
"a backslash: \\ ";
  a backslash: \
"a double quote: \" ";
  a double quote: "
```

string operators

```
"hello"."world";
  helloworld
"hello"." "."world";
  hello world
"hello"x3;
  hellohellohello
5 x 2
  55
5 * 2
  10
```

numerical operators

An operator takes some values (operands), operates on them, and produces a new value.

Numerical operators:

```
+   -   *   /
**              exponentiation
sqrt            square root
++ --           autoincrement (more later)
```

```
((1+1)**3)
   8
sqrt(64)
   8
```

automatic conversion between numbers and strings

```
5 x 2      # "5" x 2
  55
5 * 2
  10
"5" * "2"
  10
```

automatic conversion between numbers and strings

- if you use the correct operator e.g. . + *

perl will do the right thing

- if you use a weird one e.g. `"Matt" * 2`

perl can warn you:

```
Argument "Matt" isn't numeric
```

- turn on warnings using

```
#!/usr/bin/perl –w
```

in the first line

Scalar variables

```
$example_variable
$ExampleVariable
$example = "Matt";
$microsat = "AAG" x 20;
$num1 = 20**2;
$num2 = $num1 / 5;
$num2 += 5;
```

Scalar variables

```
$num ++;
```

same as: `$num = $num + 1;`

```
$num +=2;
```

same as: `$num = $num + 2;`

```
$myname = "Matt";
$myname .= " Webster\n";
print $myname;
```

```
Matt Webster
```

Operator	meaning
++	add 1
--	minus 1
+=	add a number
*=	subtract a number
.=	append a string

Tips

- Give meaningful names to variables:
  - `$dna_sequence` is better than `$n`

- Variable names in Perl are case-sensitive. This means that the following variables are different
  - `$varname = 1;`
  - `$VarName = 2;`
  - `$VARNAME = 3;`

- Perl has a long list of scalar special variables
  - `$_, $1, $2, $&`
  - avoid them!

Interpolation

- This means variables are converted to their values within double-quotes

```
$drink = "tequila";
print "Would you like a $drink?\n";
```

```
Would you like a tequila?
```

- backslash is used to prevent interpolation

```
print "Would you like a \$drink?\n";
```

```
Would you like a $drink?
```

Comparisons using `if`

- comparison operators:

Comparison	Numeric	String
equal	==	eq
not equal	!=	ne
less than	<	lt
more than	>	gt
less or equal	<=	le
more or equal	>=	ge

- how to use:

```
if ($drink eq "tequila") {
   print "Thank you for the tequila\n";
}
```

Comparisons using `if`

```
if ($drink eq "tequila") {
   print "Thank you for the tequila\n";
} else {
   print "I would prefer tequila\n";
}

if ($cost == 10) {
   $cost+=2;
   print "$cost\n";
}
12
```

don't confuse
= and ==

Boolean Values

If the value is a number. 0 is false, other numbers are true.

If the value is a string. " " is false, other strings are true.

```
$drink = "tequila";

if ($drink)          #true

if ($drink = "wine")   #true

if ($drink eq "wine")  #false
```

User input from <STDIN>

```
$line = <STDIN>;
if ($line eq "\n")  {
   print "that was just a blank line!\n";
} else {
   print "you typed $line";
}
```

`chomp` function

- `chomp` removes `\n` from the end of a line

```
$line = <STDIN>;
chomp $line;
```

or

```
chomp ($line = <STDIN>);
```

- `chomp` is very useful when reading in files

`while` control structure

- `while` keeps repeating if a condition is true

```
$i = 0;
while ($i < 10) {
   print "the count is $i\n";
   $i++;
}

while ($line=<STDIN>) {
   chomp $line
   print "$line\n";
}
```

undef

- `undef` means nothing
  - or undefined

- is automatically converted into empty string or zero if you use an undefined variable

```
$n++;
print $n;
1
```

summary

- variables in perl (e.g. `$example`) can be numbers or strings
- perl converts variable between the two depending on the operation
- backslash `\` used to indicate special characters like newline `\n` and tab `\t`
- compare numbers or strings with comparison operators and `if`
- `<STDIN>` gets input from the keyboard
- `chomp` removes `\n`
- `while` repeats a chunk of code while an expression is true