

Scalars

What are scalars?
Creating scalar variables
Operating on scalars

Slide 1

What are scalars?

- In Perl, a *scalar* is a singular piece of data, either a number or a string
- Some examples:
14
hello
5.64234
This sentence is a scalar!
<the entire text of the Programming Perl book>

Slide 2

What are scalars?

- Perl treats numbers and strings nearly interchangeably
- If you use a string where Perl expects a number, it will use the string as a number, and vice versa. Examples later.

Slide 3

Numbers

- Integers
5
745634
- Floating Point
5.2304
3.88e12
-56.3421

Slide 4

Other Number Bases

- Hexadecimal
 - 0x3A5C91
- Octal
 - 03742
- Binary
 - 0b1001011101

Slide 5

Strings

- String *literals* are represented in a Perl program within quotes
 - 'hello!'
 - 'Four score and seven years ago'
 - 'What\'s for dinner?'
- Everything within the single quotes is interpreted literally. Use `\` to represent `'` in your strings.

Slide 6

Double-Quoted Strings

- To represent *control characters* within strings, use double-quotes instead.
 - “Go Blazers!\n”
 - “Fran\tFabrizio”
 - “She said, \”Take me to your leader.\””
- There are lots of control characters and the `\` takes on a lot of power within double quotes. These are just the most common uses.

Slide 7

Other Ways to Make Strings

- Having to escape quotes that appear within strings is awkward. One alternative...
- Use `q//` and `qq//` operators
 - `print q/She said, “Take me to your leader.”/;`
 - `print qq/”Hey there.”\n”How are you?”/;`
- Alternate delimiters available, too
 - `print qq#http://www.cis.uab.edu/it/\n#`

Slide 8

Other Ways to Make Strings

- Here-Documents (“Here Docs”)
- Great for when you have a lot of output that you want to be formatted exactly (HTML, for ex.)
- ```
print <<EndHTML;
<html><head><title>New Page</title></head>
 <body>This is a new page of text.</body>
</html>
EndHTML
```

Slide 9

## Numbers <--> Strings

- Perl will convert strings to numbers when needed  

```
'35' + 4 # Equals 39
"16fran34" / " 8" # Equals 2
"tony" + 3 # Equals 3, non-numeric string
converts to 0
```
- Numbers to strings works as well  

```
"Y" . 2 . "K" # Equals "Y2K"
```

Slide 10

## Operating on Scalars

- For numbers, all the usual *operators*  

```
+ - * / % **
```
- For strings, we have
  - The concatenation operator  

```
"base" . "ball" # Equals "baseball"
```
  - The repetition operator  

```
'ho' x 3 # 'hohoho'
3 x 3 # "333" Don't get fooled :-)
```

Slide 11

## Operating on Numbers

- ```
print "14 divided by 4 is ", 14 / 4, ", that's a
remainder of ", 14 % 4, "\n";
```

 - “14 divided by 4 is 3.5, that's a remainder of 2.”
- ```
print 2**3; # prints "8"
```

Slide 12

## Truth and Falsehood

- Only five things in Perl mean “false”. They are:
  - 0
  - “0”
  - “” (empty string)
  - Undefined
  - () (Empty list)

Slide 13

## Comparing Numbers

- print “Are these equal?”, 1 == 2, “\n”;  
print “How about these?”, 3 == 3, “\n”;
  - Are these equal?  
How about these? 1
- So “true” comparisons return 1, and “false” comparisons return the empty string. Very helpful for conditional statements (next week).

Slide 14

## Comparing Numbers

- Other Comparisons
  - print “Is 2 bigger than or equal to 1?”, 2 >= 1;
    - Is 2 bigger than or equal to 1? 1
  - print “8 less than 6?”, 8 < 6;
    - 8 less than 6?

Slide 15

## The Spaceship: <=>

- The <=> operator is not strictly a comparison.
- It returns 1 if the left side is bigger, 0 if both sides are the same, and -1 if the right side is bigger.
  - print “1 and 2?”, 1 <=> 2, “\n”; # 1 and 2? -1
  - print “2 and 2?”, 2 <=> 2, “\n”; # 2 and 2? 0
  - print “3 and 2?”, 3 <=> 2, “\n”; # 3 and 2? 1

Slide 16

## Boolean Operators

- `&&` is and, `||` is or
- `print "Test one: ", 4 > 2 && 6 > 5, "\n";`
  - Test one: 1
- You can also use the words "and" and "or" instead of `&&` and `||`. However, read page 30 in the text for why this is tricky (hint: precedence)

Slide 17

## String Operators

- Concatenation with `.`
  - `print "Dogs chase " . "cats";`
- Repetition with `x`
  - `print "Cats and dogs " x 3;`
    - Cats and dogs Cats and dogs Cats and dogs
- `print "M" . "iss" x 2 . "ippi";`
  - Mississippi

Slide 18

## String Comparison

- `print "Is dog equal to dog?", "dog" eq "dog";`
  - Is dog equal to dog? 1
- `print "Are cats and dogs different?", "cat" ne "dog";`
  - Are cats and dogs different? 1
- Also have `gt`, `lt` operators, good for alphabetical ordering

Slide 19

## Precedence of Operators

- See the chart on p. 36 of the text and understand the implications of operator precedence

Slide 20

## Scalar Variables

- Perl uses the \$ to indicate a scalar *variable*  
\$name  
\$return\_value  
\$aLengthyVariableName
- Variable names are case-sensitive, can contain alphanumeric and underscore characters, cannot start with a digit, and can be of any length

Slide 21

## Scalar Assignment

- \$day = 'Wednesday';
- \$grade = '95';
- \$bonus = 5;
- \$total = \$grade + \$bonus; # 100
- \$extra\_credit = 10;
- \$total += \$extra\_credit; # 110
- \$day .= ' afternoon'; # "Wednesday afternoon"

Slide 22

## String Interpolation

- When using double-quotes, not only does the \ become more powerful, but you can also perform *variable substitution*  
\$day = "Wednesday";  
\$phrase = "\$day night"; # Wednesday night  
\$literal = '\$day night'; # \$day night

Slide 23

## String Interpolation 2

- When perl sees a \$ inside double-quotes, it tries to form the longest variable name possible  
\$word = 'dog';  
\$string = "The \$word barks."; # The dog barks  
\$string = "Cats and \$words"; # Cats and

.... because \$words is not defined. Instead:

```
$string = "Cats and ${word}s"; # Cats and dogs
```

Slide 24

## String Interpolation 3

- Four ways to accomplish the same thing

```
$word = 'dog';
$string = "Cats and ${word}s\n";
$string = "Cats and $word" . "\n";
$string = "Cats and $word" . s . "\n";
$string = "Cats and " . $word . "\n";
```

Slide 25

## Comparing Scalars

- For numbers, you can use:  
== != > < >= <=
- For strings, you can use:  
eq ne gt lt ge le
- A very common Perl mistake:  
'cat' == 'dog' # True, Perl makes both sides 0  
'cat' eq 'dog' # False, what you meant to do

Slide 26

## undef

- If a scalar variable has not yet been assigned a value, it has the Perl special value *undef*
- undef behaves like 0 when used as a number and like an empty string when used as a string  
5 + \$sum # equals 5  
\$sum++;  
5 + \$sum # equals 6  
\$newstring .= "hello"; # \$newstring is "hello"

Slide 27

## Boolean Values

- Perl does not require or have a specific boolean data type
- The following evaluate to false  
undef 0 '0' ''
- Everything else evaluates to true
- Use the ! to get the opposite of a boolean  
(! \$done) # if \$done is 1, evaluates to 0

Slide 28

## Simple output

- print “Hello World!\n”;
- \$weather = 'rainy';
- print “It will be \$weather today.\n”;
- print “My name is “;  
print “Fran”;  
print “\n”;
- print “My name is ” . “Fran” . “\n”;

Slide 29

## The if Construct

- if (*some condition*) {  
# do this if condition is true  
} else {  
# do this if condition is false  
};
- if (\$name eq 'Fran') {  
print “Hello Mr. Fabrizio\n”;  
}

Slide 30

## The if Construct 2

- if (!\$finished) { print “Keep working!\n”; }
- if (\$pushups >= 30) {  
print “Great job!\n”;  
} elsif (\$pushups >= 20) {  
print “Not too bad.\n”;  
} else {  
print “You'd better visit the new rec center.\n”;  
}

Slide 31

## Simple input

- Use <STDIN>
  - The full explanation of this will occur in a later lecture
- print “Your name please: “;  
\$name = <STDIN>;  
print “Hello \$name!”;

Slide 32

## Simple input 2

- Note that `<STDIN>` grabs an entire line of input from standard input (typically the keyboard), including the newline character at the end
- The `chomp()` function is handy here
  - `$name = <STDIN>;`  
`chomp $name;`
  - `chomp()` looks for and removes a trailing `\n`
  - Important to remember to remove it, otherwise it will mess up things like string comparisons