IT441

NETWORK SERVICES ADMINISTRATION

Perl: *Scalars*

Scalars

- What is a Scalar?
- Types of Scalars:
  - Numbers
    - Integers (...-2, -1, 0, 1, 2,...)
    - Floating-point (i.e., decimals: -1.4, 0.8, 3.7)
  - Strings
  - Logical or Boolean
- Automatic conversion between types
  - Experiment with Perl to see what happens

Numbering Systems

- Decimal

```
$num = 345;
```

- Binary

```
$num = 0b101011001;
```

- Octal

```
$num = 0531;
```

- Hexadecimal

```
$num = 0x159;
```

This is how we can express integer literals in bases *other than 10*

←

- See also: **goodnums.pl** and **badnums.pl**

Constant (Literal)

- What is a constant?
- Why do we use a constant?
- Examples of constants:

  `pi` : **3.14159265359...**

  `e` : **2.71828182845...**

  `# of states in USA`: **50**
- Write a simple program to print out a few constants.
- How is the textbook using the term "constant", versus the use in some other languages?

Quoting Strings

- Double-quoted vs. Single-quoted strings
- Double-quoted strings can be ***interpolated***, where string is subject to processing first:
  - Variables: `"My name is $name"`
  - Escaped chars: `"This is a line.\n"`
- Single-quoted strings are ***not*** interpolated, so variables, escaped characters, etc. are not recognized.
- Alternate quotes:

```
'Hello'  →   q/Hello/
"Hello"  →  qq/Hello/
```

Variables

- What is a variable?
- How do we name variables?
  - Starts with **$**
  - Next either *letter* or **_**
  - Rest can be *letters* or *numbers*
- You should develop a **pattern** so you are consistent within your programs.
- Make the name **mean something**!!!

Here Document

- Another way to write a string
- Used to input a large amount of text
- Starts with a << followed by a label

```
print << "EOF";

This is a here document
It will print exactly as shown
It is easier than quoting

EOF
```

Or...

```
$data = << "EOF";
...
```

Here Document

- Whether you surround **EOF** with single or double quotes determines whether *interpolation* takes place...

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

$num_1 = 5;
$num_2 = -8.2;
$num_3 = 7.5;


$txt = << "EOF";

Num 1 is $num_1
Num 2 is $num_2
Num 3 is $num_3

EOF


print $txt;
```

```
$ ./multiline.pl

Num 1 is 5
Num 2 is -8.2
Num 3 is 7.5
```

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

$num_1 = 5;
$num_2 = -8.2;
$num_3 = 7.5;


$txt = << 'EOF';

Num 1 is $num_1
Num 2 is $num_2
Num 3 is $num_3

EOF


print $txt;
```

```
$ ./multiline.pl

Num 1 is $num_1
Num 2 is $num_2
Num 3 is $num_3
```

Numeric Operators

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `**` Exponentiation
- `%` Modulo (Remainder)

- In arithmetic, Perl will automatically convert strings to numbers→

```
$ ./re.pl
perl>  print 2.5 * 4 ;
10

perl>  print "2.5" * 4 ;
10

perl>  print "2.5" * "4" ;
10

perl>
```

- Note: Take care to remember order of operations…

String Operators

`.` Concatenation

`ord()` ASCII Value of a character

`x` Repetition

- A number can be treated either as a number or as a string.

  In other programming languages, those would simply be separate data types – "string", "integer", "float", etc.

  In Perl, however, they are all ***scalar***

- Perl uses the *context* to decide whether the value is a number or a string

```
perl>  print 4 + 5 ;
9

perl>  print 4 . 5 ;
45
```

Booleans in Perl

- In Perl, a number of values may be considered "false": `0` , `"0"` , `""` (empty string), etc.
- We will often use `1` and `0` for *true* and *false*, respectively
- Boolean operators:
  - `&&` And
  - `||` Or
  - `!` Not
- *If you try to print a boolean value, the resulting output may appear rather odd…*

Numerical Comparisions

- `==` Equality
- `!=` **In**equality
- `<` Less Than
- `<=` Less Than or Equal To
- `>` Greater Than
- `>=` Greater Than or Equal To

`<=>` *Comparison* ("spaceship", "shuttle")

- **Left < right :** returns `-1`
- **Equal :** returns `0`
- **Left > right :** returns `1`

String Comparisions

- String comparisons are ***lexicographic***, based on the characters' numeric values – see **`http://www.asciitable.com`**

`lt`	Less Than	**`le`**	Less Than or Equal To
`gt`	Greater Than	**`ge`**	Greater Than or Equal To
`eq`	Equal To	**`ne`**	***Not*** Equal To

- **`cmp`** *Comparison*

  Left *comes before* **right :** returns **`-1`**

  Same string : returns **`0`**

  Left *comes after* **right :** returns **`1`**

Increment/Decrement Expressions

- **++$a** Pre-increment
- **--$b** Pre-decrement
- **$a++** Post-increment
- **$b--** Post-decrement

- Both will increase (or decrease) a numerical variable by ***one***
- However, in many contexts, the pre/post difference is crucial!
- It concerns two separate actions:
  - *Getting*, or reading, a variable's value (for use in an expression)
  - *Setting* , or writing, the value (to a number 1 *less* or 1 *greater*)
- ***Pre*** and ***post*** concern the order in which these things happen...

Variable Scoping in Perl

- **Scoping:** Blocks of code limit the range of a variable's definition

```
$numDef=25;
print $numDef;
      { my $numDef=1;
      print $numDef;}
print $numDef;
```

- **The my keyword:** Makes **$numDef** ***lexical*** – local to the block!
- Changes in the block will not affect the identically-named variable outside the block
- The **strict** pragma will require all variables declared with my or another keyword

Getting Data into the Program

- Use the file handle **<STDIN>**
- Try this out

```
print "Input something:";
my $newInput=<STDIN>;
print $newInput;
```

Or… **<>** **(*without* STDIN, but it behaves differently…)**

- **chomp ($newInput);** - **remove *newline* from end**
- **chop ($newInput);** - **remove *last char* from end**