Operators

Operators

- Perl has MANY operators.
  - Covered in Chapter 3 of Camel
  - **`perldoc perlop`**
- Many operators have numeric and string version
  - remember Perl will convert variable type for you.
- We will go through them in decreasing precedence.

Increment/Decrement

- **`++`** and **`--`**
  - Prefix and Postfix work as they do in C/C++
  - **`my $y = 5;  my $x = $y++;`**
    - $y → 6, $x → 5
  - **`my $y = 5;  my $x = ++$y;`**
    - $y → 6; $x → 6

Incrementation Magic

- ++ is "magical". (-- is not)
  - if value is purely numeric, works as expected
  - if string value, magic happens
  - `my $v = '99'; $v++;` → '100'
  - `my $w = 'a9'; $w++;` → 'b0'
  - `my $x = 'Az'; $x++;` → 'Ba'
  - `my $y = 'zz'; $y++;` → 'aaa'
  - `my $z = 'zZ9'; $z++;` → 'aaA0'

Even better…

- In addition to that magic, ++ will also automatically convert **`undef`** to numeric context, and then increment it.

```
#!/usr/bin/env perl
use strict;
use warnings;
my $a;
$a++;
print "$a\n";
```

- prints 1 with no warnings or errors

Exponentiation

- ** → Exponentiation.
  - works on floating points or integers
  - **`2**3`** → pow(2, 3) → "2 to the power of 3" → 8
- NOTE: higher precedence than negation
  - -2**4 → -(2**4) → -16

Unary Operators

- logical negation: !
  - **`0, '0', '', undef`** ➔ all false
  - anything else ➔ true
  - empty array in scalar context ➔ 0 ➔ false
- arithmetic negation (if numeric): -
  - if non-numeric, 'negates' the string
  - ex: **`$foo = '-abc'; $bar = -$foo;`**
  - $bar gets value '+abc'
- bitwise negation: ~

Multiplicative

- / -- Division. Done in floating point.
- % -- Modulus. Truncates operands
- * -- Numeric multiplication
- x -- String multiplication (aka repetition).
  - 123 * 3 ➔ 369
  - 123 x 3 ➔ '123123123' (scalar context)
  - (1, 2, 3) x 3 ➔ (1, 2, 3, 1, 2, 3, 1, 2, 3) (list context)

Additive

- + – normal addition
- - – normal subtraction
- . – string concatenation
  - **`$var1 = 'hello'; $var2 = 'world';`**
  - **`$var3 = $var1 . ' ' . $var2;`**
    - $var3 contains "hello world"
  - usually easier to just use interpolation:
  - **`$var3 = "$var1 $var2";`**
    - $var3 contains "hello world"

Shift operators

- << and >>
- Shift bits in left argument number of places in right argument
- **1 << 4** ➔ 16
  - 0000 0001b << 4 ➔ 0000 1000b ➔ 16d
- **32 >> 4** ➔ 2
  - 0010 0000b >> 4 ➔ 0000 0010b ➔ 2d

Relational Operators

Numeric	String	Meaning
>	**gt**	Greater Than
>=	**ge**	Greater Than or Equal
<	**lt**	Less Than
<=	**le**	Less Than or Equal

- String1 is "less than" String2 if the first differing character come first on the ASCII chart.
  - "abc" is less than "add"
  - shorter substring comes before whole string
    - "abc" is less than "abcd"

Equality Operators

Numeric	String	Meaning
==	**eq**	Equal to
!=	**ne**	not equal to
<=>	**cmp**	comparison

- About the comparison operators:
  - -1 if left < right
  - 0 if left == right
  - 1 if left > right

The danger of mixing contexts

- `my $s1 = 'Foo Bar';`
- `my $s2 = 'Hello World';`
- `if ($s1 == $s2){print "Yes\n"; }`
  - The `==` operator expects two numbers. Converts both strings to 0. `0 == 0`
- `$x = <STDIN>;  #user enters 42`
- `$y = <STDIN>;  #user enters 42.00`
- `if ($x eq $y) {print "Yes\n"; }`
  - `'42'` is not the same string as `'42.00'`

Bitwise Operators

- **&** -- AND. **|** -- OR **^** -- XOR
  - & has higher precedence
- if either value numeric:
  - convert to integer,
  - bitwise comparison on integers
- if both values strings:
  - bitwise comparison on corresponding bits from the two strings

Logical Operators

- **&&** - AND **||** - OR
  - && has higher precedence
- operate in short-circuit evaluation
  - ie, evaluate only what's needed
  - creates this common Perl line:
- `open (my $fh, '<', 'file.txt') || die "Can't open file.txt: $!";`
- return last value evaluated, not simply "true" or "false"
  - `$x = 0 || 5 || 3;`
    - $x gets value 5.
  - `$y = 5 && '' && 3;`
    - $y gets value ''
  - `$a = 0 || "" || undef;`
    - $a gets value undef
  - `$b = 5 && 3 && 1`
    - $b gets value 1

Conditional Operator

- ?: -- Trinary operator in C.
- like an if-else statement, but it's an expression
  - **`my $a = EXPR ? $b : $c;`**
  - if EXPR is true, $a = $b.
  - if EXPR is false, $a = $c
- **`my $status = $grade >= 65 ? 'pass' : 'fail';`**

Assignment operators

- **`=, **=, *=, /=, %=, x=, +=, -=, .=,`**
- **`&=, |=, ^=, <<=, >>=, &&=, ||=`**
- In all cases, all assignments of form
- **`TARGET OP= EXPR`**
- evaluate as:
- **`TARGET = TARGET OP EXPR`**

Comma Operator

- Scalar context:
  - evaluate each list element, left to right. Throw away all but last value.
  - **`$a = (fctn(), fctn2(), fctn3());`**
    - fctn() and fctn2() called, $a gets value of fctn3()
  - There is no such thing as a list in scalar context!
- List context:
  - list element separator, as in array assignment
  - **`@a = (fctn(), fctn2(), fctn3());`**
    - @a gets return values of all three functions

Logical and, or, not, xor

- Functionally equivalent to &&, ||, !
- BUT, a lower precedence.
- `$xyz = $x || $y || $z;`
- `$xyz = $x or $y or $z;`
- What's the difference?

Incomplete list

- ALL operators found in Chapter 3 of Camel.
  - And in **`perldoc perlop`**
- some skipped over, we'll talk about them later. (arrow, file test, binding)