Perl

Bjorn Freeman-Benson, June 3rd 2002

Learning a New Language

- History & Background
- Class of problems the language is intended to solve
- Data types
- Data structures
- Syntactic structures
- Semantic structures
- Appropriate algorithms
- Useful libraries
- IDE Features

History of Perl

- Larry Wall, 1987 … many others since
- Major use seems to have started in 1990
- No *a priori* design, no committees!
- There's more than one way to do it (TMTOWTDI)
- A mix-and-match accumulation of useful features desired by real programmers over many years

More Information

- **CPAN**: www.cpan.org
- www.perl.com
- www.programmingtutorials.com
- *Caveat: this talk is an abbreviation.*

Intend Uses of Perl

- Originally for text processing, generating reports
- Has features from C, Java, Unix shells, awk, and sed
- GUI front-ends to command-line commands
- Systems integration programming
- Web CGI scripting

Data Types and Structures

- Dynamically typed
- Has an "undef" value
- Numbers (integers, floats) and strings are treated the same
- "2" lt "3" ≈ 2 < 3 ≈ "2" lt 3 ≈ "2" < 3
- 0xff, 0377, 0b1101001, 12345, 12345.67, .23E-10, 4_294_967_296

Data Structures

- Scalars, arrays of scalars, associative arrays of scalars (hashes)
- $scalar number, string, reference
- @array heterogeneous
- %hash maps keys to values
- &subroutine usually omit the &
- References (not discussed)

Data Structures (Scalars)

$name = "Bjorn"

$course = 341

$course ⇒ 341

Data Structures (Arrays)

@colors = ("red","green","blue", "black");

$colors[0] ⇒ "red"

$colors[1] = "GREEN";

join(",",@colors) ⇒ "red,GREEN,blue,black"

Data Structures (Hashes)

```
%longday = (
        "Sun" => "Sunday",
        "Mon" => "Monday",
        "Tue" => "Tuesday",
        …
        "Sat" => "Saturday",  );
  $longday{"Mon"} ⇒ "Monday"
  $longday{"Sat"} = "Sabado“
```

Data Structures (Slices)

@colors[0,2] ⇒ ("red", "blue")

@colors[0..2] ⇒ ("red", "green", "blue")

@longday{"Tue", "Fri"}
  ⇒ ("Tuesday", "Friday")

@colors[1,2] = (“purple”, “gold”)

@longday{"Mon", "Wed", "Fri"} = ("CSE341", "CSE341", "CSE341")

Data Structures (List Initialization)

(@foo, @bar, &SomeSub, %glarch)

So Where Are We?

- History & Background
- Class of problems the language is intended to solve
- Data types
- Data structures
- Syntactic structures
- Semantic structures
- Appropriate algorithms
- Useful libraries
- IDE Features

Syntax (Running Perl)

- .pl and .pm files
- **`perl -w foo.pl`**

-
  ```
  #!/usr/bin/perl
  use strict;
  use warnings;
  # Print the usual greeting
  print "Hello World\n";
  ```

Syntax (Variables)

- my $name
- local @colors

Syntax (Strings)

- "It's a $adjective day to ${activity}"
- 'It\'s a $adjective day to ${activity}'

-
  ```
  my $foo = <<"EOF";
  A multi-line string
  with $adjective variable substitution.
  EOF
  ```
- my $bar = <<'EOF';

Syntax (Statements)

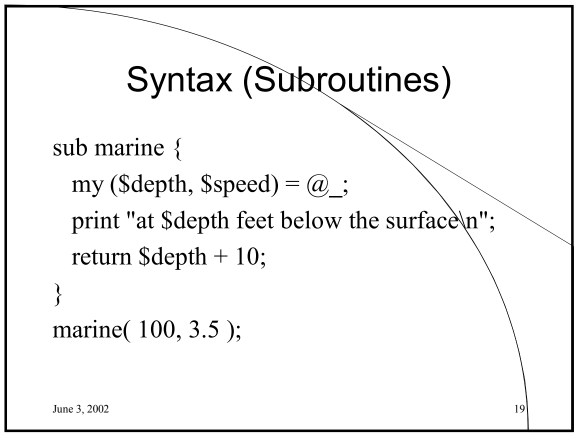
- Standard ALGOL-like…

```
if( $name eq "Bjorn" ) {
    print "Hello\n";
  }
print "Hello\n" if $name eq "Bjorn";
```

Syntax (Loops)

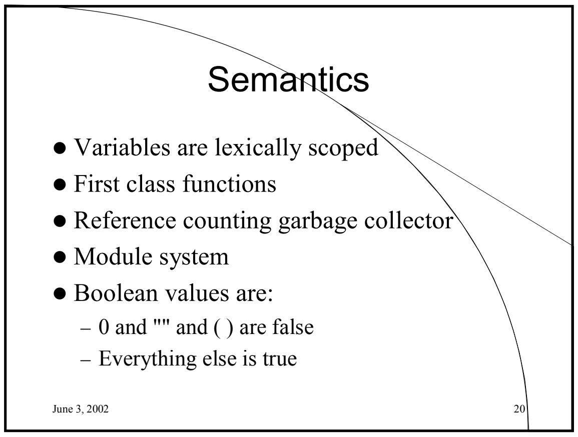
```
foreach my $color ( @colors ) {
    $allcolors = $allcolors . $color;
  }
while( … ) { … }
while( … ) { … } continue { … }
while( … ) { … next if …; … }
```

- And on and on and on…



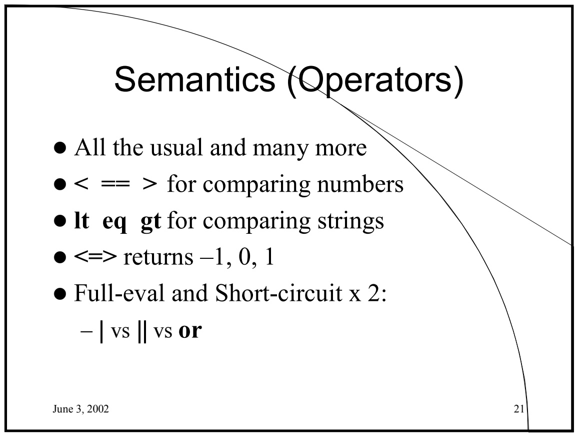
Syntax (Subroutines)

```
sub marine {
  my ($depth, $speed) = @_;
  print "at $depth feet below the surface\n";
  return $depth + 10;
}
marine( 100, 3.5 );
```



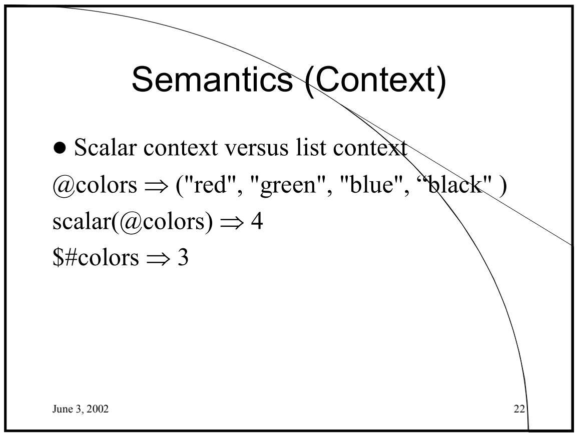
Semantics

- Variables are lexically scoped
- First class functions
- Reference counting garbage collector
- Module system
- Boolean values are:
  - 0 and "" and ( ) are false
  - Everything else is true



Semantics (Operators)

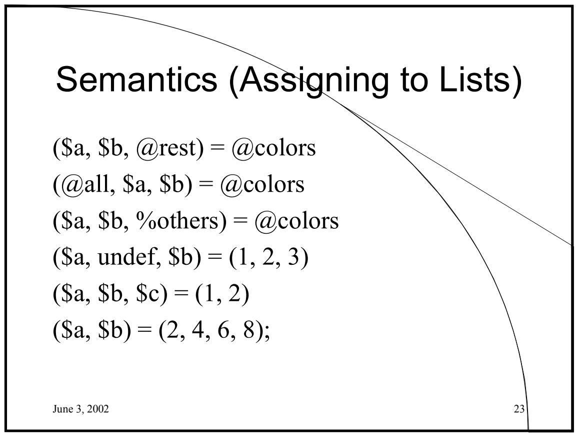
- All the usual and many more
- < == > for comparing numbers
- **lt eq gt** for comparing strings
- **<=>** returns –1, 0, 1
- Full-eval and Short-circuit x 2:
  - **|** vs **||** vs **or**



Semantics (Context)

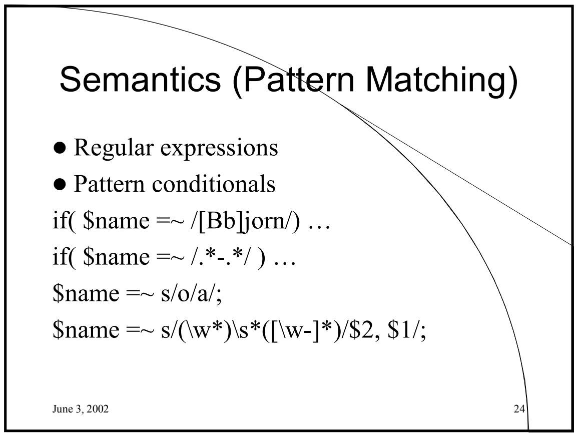
- Scalar context versus list context

@colors ⇒ ("red", "green", "blue", "black")
scalar(@colors) ⇒ 4
$#colors ⇒ 3



Semantics (Assigning to Lists)

```
($a, $b, @rest) = @colors
(@all, $a, $b) = @colors
($a, $b, %others) = @colors
($a, undef, $b) = (1, 2, 3)
($a, $b, $c) = (1, 2)
($a, $b) = (2, 4, 6, 8);
```



Semantics (Pattern Matching)

- Regular expressions
- Pattern conditionals

```
if( $name =~ /[Bb]jorn/) …
if( $name =~ /.*-.*/ ) …
$name =~ s/o/a/;
$name =~ s/(\w*)\s*([\w-]*)/$2, $1/;
```

Semantics (Predefined Variables)

● $_	Default input
● $!	Current error
● $$	Process number
● $0	Program name

- $ARGV, @ARGV
- @_ Subroutine parameters
- %SIG, %ENV, and many, many more…

So Where Are We?

- History & Background
- Class of problems the language is intended to solve
- Data types
- Data structures
- Syntactic structures
- Semantic structures
- Appropriate algorithms
- Useful libraries
- IDE Features

Libraries (File IO)

```
while( <> ) {
  print "I saw \"$_\" \n ";
}

my $line = <>;
my @alllines = <>;
```

Libraries (File IO)

```
open LIST, "files.txt" or die $!;
foreach my $filename ( <LIST> ) {
   chomp($filename); … }
close LIST;

open LIST, "ls –1 *.xml |" or die $!;
open OUTFILE, "> bar.txt";
```

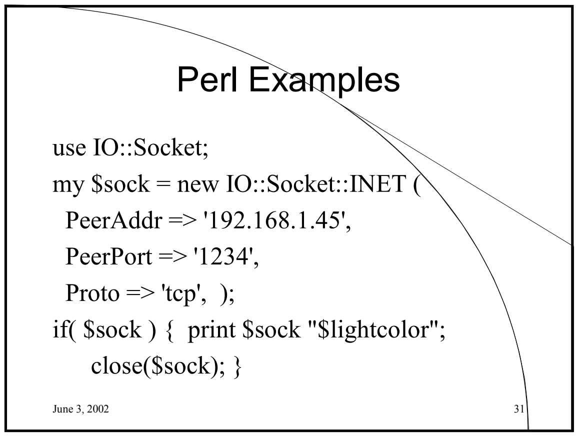
Libraries (File IO)

```
open LOGFILE, “>log.txt” or die $!;
print LOGFILE “Starting processing…\n”;
close LOGFILE;
```

Libraries (Modules)

- “use” ≈ import

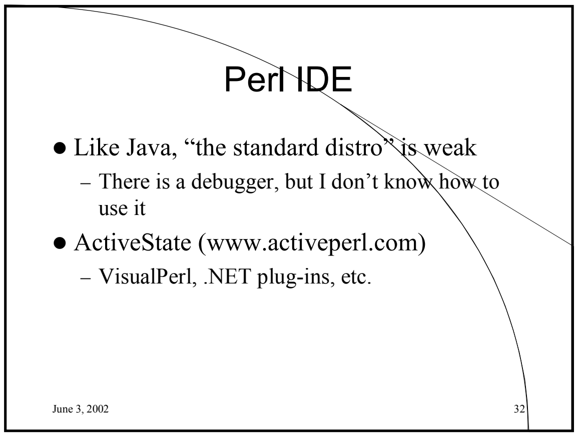
```
use File::Find;
@ARGV = qw(.) unless @ARGV;
find sub {
  print $File::Find::name, -d && '/', "\n"
}, @ARGV;
```



Perl Examples

```
use IO::Socket;
my $sock = new IO::Socket::INET (
  PeerAddr => '192.168.1.45',
  PeerPort => '1234',
  Proto => 'tcp',  );
if( $sock ) {  print $sock "$lightcolor";
     close($sock); }
```

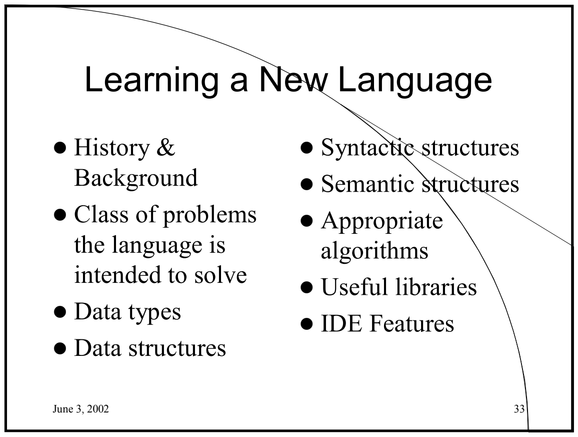
June 3, 2002 31



Perl IDE

- Like Java, “the standard distro” is weak
  - There is a debugger, but I don’t know how to use it
- ActiveState (www.activeperl.com)
  - VisualPerl, .NET plug-ins, etc.

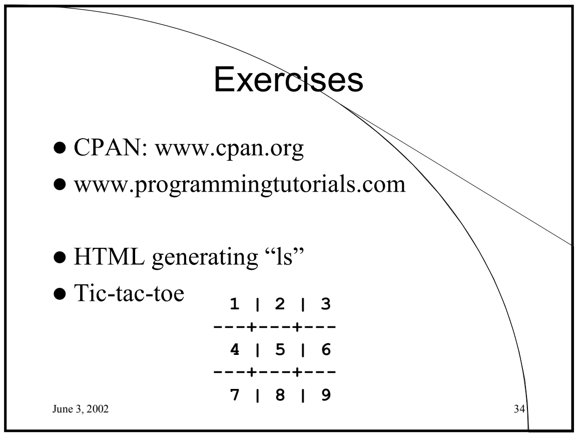
June 3, 2002 32



Learning a New Language

- History & Background
- Class of problems the language is intended to solve
- Data types
- Data structures
- Syntactic structures
- Semantic structures
- Appropriate algorithms
- Useful libraries
- IDE Features

June 3, 2002 33



Exercises

- CPAN: www.cpan.org
- www.programmingtutorials.com
- HTML generating “ls”
- Tic-tac-toe

```
 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9
```

June 3, 2002 34