Introduction to Perl

Jonathan Geisler

April 25, 2008

Perl

- Created by Larry Wall
  - Great Christian man!
  - Linguist
  - Currently recovering from stomach tumor
- TMTOWTDI is the language motto
- Portable, but grew up in UNIX
  - awk
  - sh
  - C (not C++)

References

- Perl.com
- O'Reilly Perl books are the "standard"
- `man perl`
- `perldoc function`

Datatypes

- Scalars (`$var`)
  - `5`
  - `"Hello, World!"`
  - `17.643`
- Arrays (`@var`)
  - `(5, 4, 3)`
  - `("Hello", "World")`
  - `(1.0, 2.0, 3.0, 4.0)`
- Hashes (`%var`)

Creating Variables

- Just use them! (old recommendation)
- Changed for debug-ability to
  - Declare them local with `my`
  - Declare them global with `our`
- Examples
  - `my $scalar = 1`
  - `my @array = (1, 2, 3)`
  - `my %hash = (a => '0', b => '1', c => '2')`

Arrays

- Although arrays are declared with @, a specific element is accessed as if it were a scalar with $
- 0 based (like C/C++)
- Examples
  - `$array[0]`
  - `$array[$location]`
  - `$array[$#array]`
  - `$array[-1]`

Hashes

- Also called associative arrays
- List of items accessible by non-numeric index
- Examples
  - `$dog{name}`
  - `$dog{color}`
  - `$dog{age}`
  - `$dog{$attribute}`
  - All the above are stored in %dog

I/O

- `print "Hello World!\n"`

I/O

- `print "Hello World!\n"`
- Prints “Hello World!” to stdout

I/O

- `print "Hello World!\n"`
- `$in = <>`

- Prints "Hello World!" to stdout

I/O

- `print "Hello World!\n"`
- `$in = <>`

- Prints "Hello World!" to stdout
- Reads a line from stdin

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`

- Prints "Hello World!" to stdout
- Reads a line from stdin

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`

- Prints "Hello World!" to stdout
- Reads a line from stdin
- Prints "output" to the file handle FILE

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`
- `$stuff = <INPUT>`

- Prints “Hello World!” to stdout
- Reads a line from stdin
- Prints “output” to the file handle FILE

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`
- `$stuff = <INPUT>`

- Prints “Hello World!” to stdout
- Reads a line from stdin
- Prints “output” to the file handle FILE
- Reads a line from file handle INPUT

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`
- `$stuff = <INPUT>`
- `open(FH, "filename")`

- Prints “Hello World!” to stdout
- Reads a line from stdin
- Prints “output” to the file handle FILE
- Reads a line from file handle INPUT

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`
- `$stuff = <INPUT>`
- `open(FH, "filename")`

- Prints "Hello World!" to stdout
- Reads a line from stdin
- Prints "output" to the file handle FILE
- Reads a line from file handle INPUT
- Creates file handle FH that points to filename

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`
- `$stuff = <INPUT>`
- `open(FH, "filename")`
- `close(FH)`

- Prints "Hello World!" to stdout
- Reads a line from stdin
- Prints "output" to the file handle FILE
- Reads a line from file handle INPUT
- Creates file handle FH that points to filename

I/O

- `print "Hello World!\n"`
- `$in = <>`
- `print FILE "output\n"`
- `$stuff = <INPUT>`
- `open(FH, "filename")`
- `close(FH)`

- Prints "Hello World!" to stdout
- Reads a line from stdin
- Prints "output" to the file handle FILE
- Reads a line from file handle INPUT
- Creates file handle FH that points to filename
- Closes file handle FH

Strings

- Strings are first class citizens in Perl
- `"$variable"` is interpolated
- `'$variable'` is not
- `q()`, `qw()`, and `qq()` can be used for special use cases (e.g., `qw()` is often used to initialize hashes)
- Separate string operators (`.`, `eq`, `ne`, `lt`, etc.)
- Can use regular expressions with them (covered next week)

Syntax

Looks similar to C with some changes:

- `until`/`unless`
- `foreach`
- `next`/`last`
- `elsif`
- `sub`

Lists

- Perl treats lists and arrays the same
- Powerful commands are built into the language
  - `push()/pop()`
  - `split()/join()`
  - `shift()/unshift()`

Context is important

Perl does different things at different times to achieve DWIMery:

- Arrays seen as both lists and scalars
  - `foreach (@array)`
  - `if (@array > 3)`
- Strings seen as both characters and integers
  - `print "1.0"`
  - `$i = "1.0"* 4`

References
(I mean pointers)

- If you like C/C++ pointers, then you can almost do a straight translation from & $\rightarrow$ \ and * $\rightarrow$ $
- Since `$$var[$location]` makes the two dollars signs difficult to see, you can use `$var->[$location]` instead
- Safer than C/C++ since everything is garbage collected and no null dereferences are allowed

Subroutines

```
sub name {
    #code here
}
```

- Parameters passed in `@_` by reference
- If you want copies, make them yourself with `shift()`
- Can be mixed in the middle of code
- Can be created on the fly (closures)