

**CSI605**

**perl**

# Perl Facts

- **Perl = Pathologically Eclectic Rubbish Lister**
- **Perl is highly portable across many different platforms and operating systems**
- **Perl is really designed to digest large amounts of data**
- **Perl is not a strictly interpreted language in that Perl first compiles your program into an intermediate program**
- **Perl community has a rich history of being highly helpful**
- **Two handy webpages**
  - **[www.perl.com](http://www.perl.com)**
  - **[www.perl.org](http://www.perl.org)**
- **Perl behaves as a natural language processor**

# Perl Hello World Example

- print "hello, world \n";
- To compile and run this program save it as helloworld.pl
  - perl helloworld.pl
  - perl -w helloworld.pl
    - Includes warnings

# Scalars

$\$x = 34;$

$\$pi = 3.14;$

$\$avogadros = 6.02e23;$

$\$name = "Jeff";$

$\$procalmation = "I love \$name";$

$\$y = \$x;$

$\$a = \$pi * \$r * \$r;$

$\$cwd = `pwd`$                       string output from a command

$\$exit = system("vi \$x");$

# Scaler Behavior

- Undeclared variables are initialized to either "" or 0
- We can do things like this

```
x = '23';  
print $x + 3, "\n";
```

# Arrays

- Here is one way to assign an array  
@stuges = {"curley", "moe", "larry"};
- We may alternatively do array assignment element by element
  - \$stuges[0] = "curley";
  - \$stuges[1] = "more";
  - \$stuges[2] = "larry";

# Hashes

- A hash can be thought of as an unordered set of scalars, accessed by some string value that is associated with each scalar.

- Here are two ways to declare a simple hash

```
%longdays = ("Sun", "Sunday", "Mon", "Monday",  
  "Tue", "Tuesday", "Wed", "Wednesday", "Thu",  
  "Thursday",  
"Fri", "Friday", "Sat", "Saturday");
```

```
%longdays = (  
  "Sun" => "Sunday",  
  "Mon" => "Monday",  
  "Tue" => "Tuesday",  
  "Wed" => "Wednesday",  
  "Thu" => "Thursday",  
  "Fri" => "Friday",  
  "Sat" => "Saturday"  
);
```

# Opening Files

- **STDIN** is your program's normal input channel
- **STDOUT** is your program's normal output channel
- **STDERR** is an additional output channel for your program

- **The various opens**

**open(MYPIPE, "filename");**            **#read from input file**

**open(MYPIPE, "<filename");**            **#same as above**

**open(MYPIPE, ">filename");**            **#create and write to  
file**

**open(MYPIPE, ">>filename");**            **#append to an existing  
file**

**open(MYPIPE, "| output-pipe-command");**

**#setup an output filter**

**open(MYPIPE, "input-pipe-command |");**

**#setup an input filter**

- **We close using**
  - **close(MYPIPE);**

# Reading from a File and Standard In

- Simple example

print STDOUT “Enter a number: “;

\$number = <STDIN>;

print STDOUT “The number is \$number\n”;

- This program does not handle removing the new line from your typed input
- chop and chomp;
  - chop will remove and return the last character passed to it (regardless of what it is)
  - chomp will only remove the end of record marker
- chop example  
chop(\$number = <STDIN>;
- alternative chop example  
\$number = <STDIN>;  
chop(\$number);

# Arithmetic Operators

- **Some Example Arithmetic Operators**

**$\$a + \$b$**

**$\$a * \$b$**

**$\$a \% \$b$**

**$\$a ** \$b$**

- **Perl provides support for the usual set of mathematical functions such as sine, cosine etc**
  - **The O'Reilly *Programming Perl* book contains a nice alphabetical lists of the functions**

# String Operators

- **String concatenation is handled by the Perl . operator**

- **Example**

```
$a = 345;
```

```
$b = 655;
```

```
print $a + $b; #prints 1000
```

```
print $a . $b; #prints 345655
```

- **There is also a string repeat or times operator**

- **Example**

```
$a = 123;
```

```
$b = 2;
```

```
print $a * $b;          #prints 245
```

```
print $a x $b;          #prints 123123
```

- **Three ways to handle output**

```
print $a . ' is equal to ' . $c . "\n";
```

- **Uses the dot operator**

```
print $a, ' is equal to ', $b, "\n";
```

- **Uses list construct**

```
print "$a is equal to $b\n";
```

- **Uses interpolation**

# Assignment Operators

- **Simple assignments**

$\$a = \$c;$

$\$a = \$c + 3;$

- **Assignment shortcut**

- $\$a = \$a * 3;$

- $\$a *=3;$

- **This of course generalizes across most of the binary operators**

# **Autoincrement and Autodecrement Operators**

- **Like C Perl supports the standard autoincrement and autodecrement operators**
- **++\$a, \$a++**
- **--\$a, \$a--**

# Logical Operators

- **Perl supports the standard set of logical operators**
  - **`$a && $b`**
  - **`$a || $b`**
  - **`!$a`**
  - **`$a and $b`**
  - **`$a or $b`**
  - **`not $a`**

# Comparison Operators

- **Numeric Comparison Operators**
  - `==`
  - `!-`
  - `<`
  - `>`
  - `<=`
  - `<=>`      **0 if equal, 1 if \$a greater, -1 if \$b greater**
- **String Comparison**
  - `eq`
  - `ne`
  - `lt`
  - `gt`
  - `le`
  - `cmp`

# File Test Operators

- **-e \$a**                **exists**
- **-r \$a**                **readable**
- **-w \$a**                **writable**
- **-d \$a**                **directory**
- **-f \$a**                **file**
- **-T \$a**                **text file**

# Nature of Truth in Perl

- Any string is true except for "" and "0"
- Any number is true except for 0
- Any reference is true
- Any undefined value is false.
- **Examples**
  - 0                    string 0 hence false
  - 1                    string 1 hence true
  - 10-10                0 and converted to string 0 hence false
  - 0.00                same as above
  - ""                    null string hence false
  - "0.00"              the string "0.00" true since neither empty or exactly "0"
  - "0.00" + 0        number 0 due to + coercion hence false
  - \ \$a                a reference to a hence true  
even if \$a is false

# Conditionals

- **Perl supports the usual constructs**

```
if($state eq "Virginia"){
    print "Southerner\n";
}
elsif ($state eq "New York"){
    print "Yankee\n";
}
else{
    print "Martian\n";
}
```

- **Unless construct**

```
unless ($name eq "Jeff"){
    print "It's not me. \n";
}
```

## for example

```
for ($sold = 0; $sold < 500; $sold += $purchase){  
  
    $available = 500 - $sold;  
    print "$available tickets are available. How many  
    would you like:";  
    $purchase = <STDIN>;  
    chomp($purchase);  
  
}
```

## while example

```
while ($tickets_sold < 500){  
  
    $available = 500 - $tickets_sold;  
    print "$available tickets are available. How  
    many would you like:";  
    $purchase = <STDIN>;  
    chomp($purchase);  
    $tickets_sold += $purchase;  
  
}
```

# Average Grades Example

```
open(GRADES, "grades") or die "Can't open grades:  
    $!\n";  
while ($line = <GRADES>) {  
    ($student, $grade) = split(" ", $line);  
    $grades{$student} .= $grade . " ";  
}  
  
foreach $student (sort keys %grades) {  
    $scores = 0;  
    $total = 0;  
    @grades = split(" ", $grades{$student});  
    foreach $grade (@grades) {  
        $total += $grade;  
        $scores++;  
    }  
  
    $average = $total / $scores;  
    print "$student: $grades{$student}\tAverage:  
    $average\n";  
  
}
```

# Debugging in Perl

- **The first thing to note is we can ascertain a lot about our Perl code by executing it with the -w switch**
- **We can launch the perl debugger by invoking the -d option**
- **The debugger supports**
  - **breakpoints**
  - **single step**
  - **next**
  - **listing code**
  - **trace**

# Recommended References

- ***Learning Perl* (2nd Edition), R. L. Schwartz, T. Christiansen, and L. Wall, ISBN 1565922840, O'Reilly and Associates**
  - Not as in-depth as the *Programming Perl* Reference
- ***Programming Perl* (2nd Edition), L. Wall, T. Christiansen, and R. L. Schwartz, ISBN 1-56592-149-6**
  - Definitive Reference