COMP284 Scripting Languages
Lecture 6: Perl (Part 5)
Handouts

Department of Computer
School of Electrical Engineering, Electronics, an
University of Liverpo

Contents

Substitutions

s/*regexpr*/*replacement*/

- Searches a variable for a match for *regexpr*, and if found, replaces that match with a string specified by *replacement*
- In both scalar context and list context returns the number of substi
- If no vari ~ or !~, the spe
- The binding operator !~ only negates the re affect the manipulation of the text

The delimiter / can be replaced by some other pair character, for example:

s!*regexpr*!*replacement*! or s<*regexpr*>[*replacement*]

Substitutions

Example:

```
$text = "http://www.myorg.co.uk/info/refund/../vat.html";
$text =~ s!/[^/]+/\.\.!!;
print "$text\n";
```

Output:

```
http://ww
```

Example

```
$_ = "Yabba dabba doo";
s/bb/dd/;
print $_, "\n";
```

Output:

```
Yadda dabba doo
```

Note: Only the first match is replaced

Substitutions: Capture variables

s/*regexpr*/*replacement*/

- Perl treats *replacement* like a double-quoted string
  - ⇝ backslash escapes work as in a double-quoted string

\	
\	
\	
\	
\u	Upper case next letter
\U	Upper case all following let

  - ⇝ variable interpolation is applied, including ca

$*N*	string matched by capture group *N* (where *N* is a natural number)
$+{*name*}	string matched by a named capture group

Substitutions: Capture variables

Example:

```
$name = "Dr␣Ullrich␣Hustadt";
$name =~ s/(Mr|Ms|Mrs|Dr)?\s*(\w+)\s+(\w+)/\U$3\E, $2/;
print "$name\n";

$name = "Dave␣
$name =
print "$name
```

Output:

```
HUSTADT, Ullrich
SHIELD, Dave
```

Substitutions: Modifiers

Modifiers for substitutions include the following:

s/ / /g	Match and replace globally, that is, all occurrences
s/ / /i	Case-insensitive pattern matching
s/ / /m	Treat string as multiple lines
s/ / /s	
s/ / /e	

Combin

Example:


```
$_ = "Yabba dabba doo";
s/bb/dd/g;
print $_,"\n";
```

Output:

```
Yadda dadda doo
```

Substitutions: Modifiers

Modifiers for substitutions include the following:

s/ / ./e	Evaluate the right side as an expression

Example:

```
1 $text = "The
2 $text =
3
4 print "$t
5 $text =~ s!(\d+\.\d+)!sprintf("%d",$1+0.5)!
6 print "$text\n";
```


Output:


```
The temperature is 40.5555555555556 degrees Celsius
The temperature is 41 degrees Celsius
```

Regular Expressions and the Chomsky Hierarchy

- In Computer Science, formal languages are categorised according to the type of grammar needed to generate them (or the type of a recog
- Perl re at least recognise all context-free languages
- However, this does not mean regular expression s parsing context-free languages
- Instead there are packages specifically for parsing context-free languages or dealing with specific languages, e.g. HTML, CSV

Java methods versus Perl subroutines

- Java uses methods as a means to encapsulate sequences of instructions
- In Java you are expected
  - to declare the type of the return value of a method
  - to provide a list of parameters, each with a distinct name, and t


```
public st
  f = f+s;
  return f;
}

public static void main(String[] args){
  System.out.println("Sum of 3 and 4 is " + sum2(3, 4)
}
```


- Instead of methods, Perl uses subroutines

Subroutines

Subroutines are defined as follows in Perl:

```
sub identifier {
  statements
}
```


- Subro
  shoul
- All sub
  - The statement
    `return value`
    can be used to terminate the execution of a subroutine a
    to make `value` the return value of the subroutin
  - If the execution of a subroutine terminates without encountering a `return` statement, then the value of the last evaluation of an expression in the subroutine is returned

The return value does not have to be scalar value, but can be a list

Parameters and Arguments

Subroutines are defined as follows in Perl:

```
sub identifier {
    statements
}
```


- In Perl t
  (or thei
  ⇝ th
- Arguments are passed to a subroutine via a special ar
- Individual arguments are accessed using
- Is is up to the subroutine to process arguments as is app
- The array `@_` is private to the subroutine
  ⇝ each nested subroutine call gets its own `@_` array

Parameters and Arguments: Examples

- The Java method

```
public static int sum2( int f, int s) {
  f = f+s;
  return f;
}
```

could b

```
sub sum2 {
  ret
}
```

- A more general solution, taking into account that a s
given arbitrarily many arguments is the following

```
1 sub sum {
2   return undef if (@_ < 1);
3   $sum = shift(@_);
4   foreach (@_) { $sum += $_ }
5   return $sum;
6 }
```

Private variables

```
sub sum {
  return undef if (@_ < 1);
  $sum = shift(@_);
  foreach (@_) { $sum += $_ }
  return $sum;
}
```


The varia


```
$sum = 5;
print "Value
print "Return value of sum: ",&sum(5,4,3,2,1),"\n";
print "Value of \$sum after  call of sum: ",$sum,"\n"
```


produces the output

```
Value of $sum before call of sum: 5
Return value of sum: 15
Value of $sum after  call of sum: 15
```

This use of global variables in subroutines is often undesirable

↝ we want $sum to be private/local to the subroutine

Private variables

- The operator my declares a variable or list of variables to be private:

```
my $variable;
my ($variable1,$variable2);
my @array;
```


- Such a declaration can be combined with a (list) assignment:

```
my va
my (v
my @a
```


- Each call of a subroutine will get its own copy of its privat

Example:


```
sub sum {
  return undef if (@_ < 1);
  my $sum = shift(@_);
  foreach (@_) { $sum += $_ }
  return $sum;
}
```

Calling a subroutine

A subroutine is called by using the subroutine name with an ampersand & in front possibly followed by a list of arguments

The ampersand is optional if a list of arguments is present


```
sub identifier {
   statements
}

... &id
... &id
... identifier(arguments) ...
```


Examples:


```
print "sum0: ",&sum,"\n";
print "sum0: ",&sum(),"\n";
print "sum1: ",&sum(5),"\n";
print "sum2: ",&sum(5,4),"\n";
print "sum5: ",&sum(5,4,3,2,1),"\n";
$total = &sum(9,8,7,6)+&sum(5,4,3,2,1);
&sum(1,2,3,4);
```

Persistent variables

- Private variables within a subroutine are forgotten once a call of the subroutine is completed
- In Perl 5.10 and later versions, we can make a variable both private and persistent using the `state` operator
  - ⇝ th
    in

Example

```
use 5.010;

sub running_sum {
  state $sum;
  foreach (@_) { $sum += $_ }
  return $sum;
}
```

Persistent variables

Example:

```
1  use 5.010;
2
3  sub running_sum {
4    state $sum;
5    fore
6    retu
7  }
8
9  print "running_sum():\t\t",    running_sum(),      "\n";
10 print "running_sum(5):\t",     running_sum(5),     "\n";
11 print "running_sum(5,4):\t",   running_sum(5,4),   "\n";
12 print "running_sum(3,2,1):\t", running_sum(3,2,1), "\n";
```


Output:

```
running_sum():
running_sum(5):          5
running_sum(5,4):        14
running_sum(3,2,1):      20
```

Nested subroutine definitions

- Perl allows nested subroutine definitions (unlike C or Java)

```
sub outer_sub {
    sub inner_sub { ... }
}
```

- Norm
  ⇝ th i
  ins
- However, Perl allows inner subroutines to be calle
  (within the package in which they are defined)

```
sub outer_sub {
    sub inner_sub { ... }
}
&inner_sub();
```

Nested subroutine definitions

If an inner subroutine uses a local variable of an outer subroutine, then it refers to the instance of that local variable created the first time the outer subroutine was called.

Example:

```
sub outer {
  my $x = $_[0];
  sub inn
  return inner();           # ret
}
print "1: ",&outer(10),"\n";
print "2: ",&outer(20),"\n";
```


Output:

```
1: 10
2: 10 # not 20!
```

Nested subroutine definitions: Example

```
sub sqrt2 {
  my $x = shift(@_);
  my $precision = 0.001;
  sub sqrtIter {
    my ($guess,$x) = @_;
    if (is

    } else { sq

  sub improveGuess {
    my ($guess,$x) = @_;
    return ($guess + $x / $guess) / 2; }

  sub isGoodEnough {
    my ($guess,$x) = @_;
    return (abs($guess * $guess - $x) < $precision); }

  return sqrtIter(1.0,$x);
}
```

Revision

Read

- Chapter 9: Processing Text with Regular Expressions

- Cha

of

R. L. Sch

Learning Perl.

O'Reilly, 2011.

- `http://perldoc.perl.org/perlsub.html`