Usenet Gems

Brian McCauley
<nobull@mail.com>

University Hospital, Birmingham,
NHS Trust

(not here in any official capacity)

(not giving the talk in the proceedings)

Why Gems?

- Small sparkling objects of high value
- Found by sifting a mass of valueless material
  - FAQs
  - Failures to RTFM
  - Off-topic threads
  - Trolls defending the above
    - Irate netizens trying to enlighten the trolls

What are these gems?

- Seemingly simple questions
  - Often asked by newbies
- Not application area specific
- Reveal something about Perl
  - a high “oooooh” factor
  - maybe an “argh!!!” factor

substr() as subroutine argument

- Consider

```
sub stripws {
    $_[0] =~ s/\s//g;
    return $_[0];
}

$_="field1      field2 field3";
my $x = stripws(substr($_,10,10));
```

- Would expect $x='field2'
- In fact $x='field2fiel'

substr() as subroutine argument

- The elements of @_ are
  - *aliases* to the arguments
  - not *copies*

```
sub foo {
  $_[0] = 'Cooked';
}

my $q='Raw';
foo($q);
print "$q\n"; # Prints 'Cooked'
```

substr() as subroutine argument

- `substr()` is an *lvalue* function
- Assign to it directly

```
substr($s,2,2)='xxx';
```

- Or though an alias

```
$_='xxx' for substr($s,2,2);
```

- Alias has “substr magic”
- `ref()` reports type of such as LVALUE

```
print ref \substr($s,2,2);
print ref \$_ for substr($s,2,2);
```

substr() as subroutine argument

- In conclusion

```
my $s='xxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
for my $x ( substr($s,10,10) ) {
  $s = '0123456789Wierd, eh??';
  print "$x\n"; # Prints 'Wierd, eh?';
  $x= 'Just totally crazy';
  print "$s\n"; # Prints '0123456789Just totally crazy?'

  $s = 'field1       field2 field3';
  $x =~ s/\s//g;
  print "$x\n"; # Prints 'field2fiel'
}
```

Minimal matching

- Given a string
  ```
  $_ = 'qwertayuiopasdfooghjkl';
  ```
- Extract the portion ending at `'foo'` and starting at the previous `'a'`

Minimal matching

- Obvious answer

```
my ($match) = /(a.*?foo)/;
```

- Does not work
  - Non greedy qualifer does not trump first-match rule

Minimal matching

- Special case because 'a' is a single character
  ```
  my ($match) = /(a[^a]*foo)/;
  ```
  - Does not generalise to 'a' being an arbitrary pattern
- Special case for last 'foo'
  ```
  my ($match) = /.*(a.*foo)/;
  ```
  - Does not generalise to finding each 'foo'

Minimal matching

- The original poster wanted to find
  - each /$end/ in turn
  - extract from last preceding /$start/
- Want a way to anchor a regex relative to where previous search left off.
  - The \G assertion

Minimal matching

- Putting it together

```
$_ = 'axaxfoo ayayfoo';
my $start = qr/a/;
my $end = qr/foo/;
my @matches;

while ( /$end/g ) {
    push @matches => $1 if /.*($start.*$end\G)/;
}
```

The API of import

- What is the API of import()
- It's up to the module author!
  - There are only conventions
  - Some people find this distressing
- For compatability with older Perl may want import() to simulate the VERSION method
  - Treat first argument as a minimum required version

defined() and autoloaded functions

- Why is defined() is false for Fcntl constants?

```
use Fcntl;
print 0+defined(&O_APPEND),"\n"; # 0
print O_APPEND,"\n"; # prints 8
```

- It's because &O_APPEND is autoloaded
  - comes into being on first call it
  - `exists(&O_APPEND)` always true
  - `defined(&O_APPEND)` is false until call
- Or is it?

defined() and autoloaded functions

- OK so why is it

```
use Fcntl;
print O_APPEND,"\n"; # 8
print 0+defined(&O_APPEND),"\n"; # 0
```

- Well it's a bug
- But what's going on?

defined() and autoloaded functions

- Exporter inserts a CODE reference info a glob
  ```
  *O_APPEND = \&Fcntl::O_APPEND;
  ```
- But &Fcntl::O_APPEND is not defined!
- What does it mean to make a reference to an undefined function?
- it's almost like a symref

defined() and autoloaded functions

- How we expect CODErefs to work

```
sub one { 'one' };
sub foo { 'zero' };

my $bar = \&foo;
print \&foo,$bar; # Prints the same thing twice

*foo = \&one; # Emits redefined warning
print $bar->(); # Prints zero
print \&foo,$bar; # Prints different things

eval "sub foo { 'two' }"; # Emits redefined warning
print $bar->(); # Still prints zero

print 0+defined(&$bar); # Prints 1
```

defined() and autoloaded functions

- How CODErefs to undefined functions work

```
sub one { 'one' };

my $bar = \&foo; # &foo does not yet exist
print \&foo,$bar; # Prints the same thing twice

*foo = \&one;
print \&foo,$bar; # Prints different things
print $bar->(); # Prints one

eval "sub foo { 'two' }"; # Emits redefined warning
print $bar->(); # Prints two

print 0+defined(&$bar); # Prints 0
```

if something = this or that

- A programming newbie writes
```
next if $_ eq ( 'Fred' or 'Wilma' );
```
- Clearly missundersood what “or” means in a programming language
- The semantics the newbie expects of “or” are not something a programmer would expect
- But Perl *has* something with the semantics the newbie would expect of “or”
```
use Quantum::Superpositions;
next if $_ eq any('Fred','Wilma');
```

Finding all matches

- Given a string
  ```
  $_ = 'a78b9c';
  ```
- And a pattern
  ```
  my $p = qr/\d+/;
  ```
- Find the start and end of all matches
  - For convenience, express as
    - Offset of first character of match
    - Offset of first character beyond match

Finding all matches

- The simple answer
  - Scalar m//g iterator
  - The @- and @+ special variables

```
$_ = 'a78b9c';
my $p = qr/\d+/;
my @matches;

while (/($p)/g) {
  push @matches => [ $-[1], $+[1] ];
}
```

Finds '78' and '9' i.e. @matches=([1,3],[4,5])

Finding all matches

- Overlapping matches
  - Look for a zero-width target

```
$_ = 'a78b9c';
my $p = qr/\d+/;
my @matches;

while (/(?=($p))/g) {
  push @matches => [ $-[1], $+[1] ];
}
```

Finds '78', '8' and '9' i.e. @matches=([1,3],[2,3],[4,5])

Finding all matches

- Multiple matches at same start
  - Usually finds only 'best'
- Fool RE engine into backtracking
  - “always false” assertion `(?!)`
- Save values before backtracking
  - Embed code `(?{ ... })`

Finding all matches

- Putting it together

```
$_ = 'a78b9c';
my $p = qr/\d+/;
my @matches;

my $save =qr/(?{ push @matches => [ $-[1], $+[1] ] })/;

/($p)$save(?!)/;
```

Finds '78','7','8' and '9' i.e. @matches=([1,3],[1,2],[2,3],[4,5])

Thankyou

- Sildes will be available on
  - http://birmingham.pm/