On Topic

Exploring "topic" and "topicalizers" in Perl 6

Allison Randal
University of Portland
YAPC::NA 2002

On Topic

- "topic" and "topicalizer"
- This is not Vogon poetry.
- It's linguistics…
- …which might be worse.
- Be brave.

Topic in Linguistics

- Every discourse has a topic.
- Topic is "what we're talking about".
- The topic of this talk is "Topic".

Topic in Linguistics

- If you overheard:

  "I saw Lister yesterday."
  "Really? What's he up to these days?"
  "Oh, you know, drunk again, and mooning over that awful Krissy Kochanski. "
  etc., ad nauseum...

- You would know the topic was "Lister".

Topicalizers in Linguistics

- A topicalizer flags the current topic.
- Some topicalizers in English:

For our first trick tonight, ladies and gentlemen, my partner Kryten will attempt to eat a boiled egg.

Given that God is infinite, and that the universe is also infinite, would you like a toasted tea-cake?

Regarding topicalizers, I should point out that this sentence starts with one.

Topic in Perl

- Topic is the most important variable in a block of code.
- Really, the underlying data structure is the topic, not the variable.

Topic in Perl

- Variables are just names for storage locations.

- Multiple variables may be aliased to the same storage location.

    "Rimmer", "he", "the hologram", "Smeghead"

    ```
    $_, $name, %characters{'title'}
    ```

- If the topic has more than one alias, all are the current topic.

Topic in Perl

- Why learn about topic?
- It's not required.
- The first law of topic: "Topic *is* `$_`".

Topic in Perl

- To use topic, use `$_`:

```
print;
chomp;
s///;
when condition { … }
.method_call;
```

- You don't have to understand topic, but you might want to.

Topicalizers in Perl

- A topicalizer flags the current topic.
- A quick summary of topicalizers:

`given`	bare closures
`for`	`=~`
`->`	etc...
`method`	
`rule`	
`CATCH`	

Coal and Switches

- The simplest topicalizer is `given`.

```
given $name {
   when "Lister" {
      print "Smeg!";
   }
   when "Cat" {
      print "Orange?! With this suit?!";
   }
   when "Rimmer" {
      print "4,691 irradiated haggis.";
   }
}
```

Fruit Loops and M&M's

- The classic topicalizer is `for`.

```
for @orders {
   when /scone/ {
      print "Would you like some toast?"
   }
   when /croissant/ {
      print "Hot, buttered, scrummy toast?"
   }
   when /toast/ {
      print "Really? How about a muffin?"
   }
}
```

To `rw` or not to `rw`...

- In this simple form both `for` and `given` create `$_` as `rw`.

```
for @names {
   chomp;
   s:w/Arnold J\.//;
   s:w/Dave //;
}
```

Bow and Arrow

- The most flexible topicalizer is `->`.
- By itself, it defines an anonymous sub:

```
$cleanup = -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}

$intro = "Fear not, I'm Captain Rimmer!";
$cleanup($intro);
```

To `rw` or not to `rw`... (cont.)

- `->` creates its aliases read-only.
- Unless `rw` is specified.

```
$cleanup = -> $line is rw {
   s:w/Captain Rimmer!/the bloke/;
   $line _= " who cleans the soup machine!";
   print;
}
```

Bow and Arrow

- Combined with another topicalizer, `->` creates a named alias for the current topic:

```
for @lines -> $line is rw {
   s:w/Captain Rimmer!/the bloke/;
   $line _= " who cleans the soup machine!";
   print;
}
#Perl 5
for $line (@lines) {
  $line =~ s/Captain Rimmer!/the bloke/;
  $line .= " who cleans the soup machine!";
  print $line;
}
```

Bow and Arrow

- Compare:

```
for @lines -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}

$cleanup = -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}
```

Bow and Arrow

- The arrow allows certain non-topicalizers to act as topicalizers:

```
if %people{$name}{'details'}{'age'} -> $age {
   print "$age already?\n";
   if ($age > 3000000) {
      print "How was stasis?\n";
   } elsif ($age < 10) {
     print "How 'bout a muffin?\n";
   }
}
```

Bow and Arrow

- This will also work with `while`:

```
while get_next_pattern() -> $pat {
    print grep /<$pat>/, @words;
}
```

Bow and Arrow

- This feature isn't useful with all truth tests:

```
if $counter > 3 -> $value {
   # do something with $value
}

if $counter > 3 {
   my $value = 1;
   # do something with $value
}
```

Bow and Arrow

- In goofier moments -> is also called "pointy sub".
- So, remember:

Oh pointy sub, oh pointy, pointy,
Anoint this variable, anointy, nointy.

– with apologies to Steve Martin

Method in My Madness

- Methods topicalize their invocant.

```
method sub_ether ($self: $message) {
    .transmit( .encode($message) );
}

method sub_ether {
    .transmit( .encoded_message );
}

method sub_ether ( : $message) {
    .transmit( .encode($message) );
}
```

The Sub of All Fears

- Subs are not topicalizers.

```
sub eddy ($space, $time) {
    print;
}
```

- But using the `is given` property will provide the same behavior.

```
sub eddy ($space is given, $time) {
    print;
}
```

Perl Rules!

- Grammar rules topicalize their state object.

```
rule lifeform {
   <gelf> | <human> | <mechanic> | <cat>
}
```

The `CATCH`-er in the Trye

- `CATCH` blocks always topicalize `$!`.

```
CATCH {
   when Err::WrongUniverse {
      try_new_universe();
   }
}
```

The Bare Truth

- Bare closures topicalize their first argument.

```
%commands = (
   add  => { $^a + $^b },
   incr => { $_  + 1 },
);
```

Get Smart... Match

- `=~` topicalizes the variable it binds to the match.

```
s/Kryten/Holly/;

$name =~ s/Kryten/Holly/;
```

Feeling a Bit Greppish?

- `grep`-like constructs with a block.

```
@names = map { chomp; split; } @input;
```

- `grep`-like constructs without a block.

```
@names = grep /<[A-Z]><alpha>+/, @input;
```

Nesting Instinct

- Nested topicalizers add some complication.

```
for @names {
   when /Rimmer/ {
      s/Arnold\s+//;
      print;
      print rimmer_quote();
   }
   when /Kryten/ {
      for kryten_quotes() -> $quote {
         print;
      }
   }
}
```

Nesting Instinct

- There is only one topic at a time.
- Topic obeys the lexical scope of topicalizers.

```
...
when /Kryten/ {
   for kryten_quotes() -> $quote {
      print;
   }
}
...
```

Nesting Instinct

- There is only one topic at a time.
- Topic obeys the lexical scope of topicalizers.
- To keep an outer topic, use a named alias.

```
for @names -> $name {
    when /Kryten/ {
        for kryten_quotes() -> $quote {
            print $name;
            print;
        }
    }
}
```

Nesting Instinct

- Nested topicalizers within methods obscure `.methodname` calls.

```
method locate ($self, *@characters) {
    .cleanup_names(@characters);
    for @characters -> $name {
        .display_location($name);
    }
    .change_location('Holly');
}
```

Multiple Aliases

- Topicalizers aren't limited to a single alias.

```
for @characters -> $role1, $role2, $role3 {
    ...
}

for @humans, @gelfs -> $role1, $role2 {
    ...
}

for @characters; @locations -> $name; $place {
    ...
}
```

Multiple Aliases

- But the topic is consistent in each case.
- There is only one topic.
- The topic is always the first parameter.

Multiple Aliases

- The `is given` (or `is topic`) property may change which parameter becomes the topic.

```
for @characters -> $role1, $role2 is given {
    ...
}
```

The Two Minute Talk

- First Law of Topic: "Topic *is* `$_`".
- Second Law of Topic: There is only one topic.
- Third Law of Topic: When in doubt, make a named alias.

The Two Minute Talk

- Isn't that easy?