OH HAI
A bit about me...
A bit about me...in regexes
A bit about me...in regexes

/ <i>??IRL</i> Jonathan | <i>??IRC</i> jnthn /
A bit about me...in regexes

/ <?IRL> Jonathan | <?IRC> jnthon /

/ 'From ' [UK|England|Yorkshire] /
A bit about me...in regexes

/ <?IRL> Jonathan | <?IRC> jnthn /

/ 'From ' [UK|England|Yorkshire] /

/ 'Lived in ' S[lovakia&weden&pain] /
A bit about me...in regexes

/ <?IRL> Jonathan | <?IRC> jnthn /

/ 'From ' [UK|England|Yorkshire] /

/ 'Lived in ' S[lovakia&weden&pain] /

/ 'Like ' be**2r /
Perl 6 Grammars
Take regexes and...

Make it possible to write re-usable, named regexes that can call other ones – even recursively

Put them in a kind of class, so that we can use subclassing to create derived languages

Automatically build a tree data structure of the various matches
Perl 6
Grammars are wonderful 😊
But...
Input
Input → Grammar
Input → Grammar → ?
Input → Grammar → YES
Input \rightarrow \text{Grammar} \rightarrow \text{NO}

Why?
A fail grammar
We need to parse a list of destinations that a travel company sold trips to, along with the number of trips and map location.

Norway

Oslo : 59.914289, 10.738739 : 2
Bergen : 60.388533, 5.331856 : 4

Ukraine

Kiev : 50.456001, 30.50384 : 3

Switzerland

Wengen : 46.608265, 7.922065 : 3

...
Writing the grammar

Start out with an empty grammar

```java
grammar SalesExport {
}
```
Writing the grammar

The whole string will have many countries

grammar SalesExport {
    token TOP { ^ <country>+ $ }
}

A country has a name...

```java
grammar SalesExport {
    token TOP { ^ <country>+ $ }
    token country {
        <name> \n
    }
}
```
Writing the grammar

...and at least one destination.

```plaintext
grammar SalesExport {
    token TOP { ^ <country>+ $ }
    token country {
        <name> \n
        <destination>+ 
    }
}
```
Writing the grammar

A destination has a name, then a colon...

grammar SalesExport {
    token TOP { ^ <country>+ $ }
    token country {
        <name> \n
        <destination>+ 
    }
    token destination {
        \t <name> \s+ ':' \s+
    }
}
Writing the grammar

...and a latitude and longitude...

```plaintext
grammar SalesExport {
    token TOP { ^ <country>+ $ }
    token country {
        <name> 
        <destination>+
    }
    token destination {
        \t <name> \s+ ':' \s+
        <lat=.num> ',' <long=.num> \s+ ':' \s+
    }
}
```
Writing the grammar

...and a sales count.

grammar SalesExport {
    token TOP { ^ <country>+ $ }
    token country {
        <name> \n
        <destination>+
    }
    token destination {
        \t <name> \s+ ':' \s+
        \t <lat=.num> ',' <long=.num> \s+ ':' \s+
        <sales=.integer> \n
    }
}
Writing the grammar

A token to parse place names

```plaintext
grammar SalesExport {
    token TOP { ^ <country>+ $ }
    token country {
        <name> 
        <destination>+ 
    }
    token destination {
        \t <name> \s+ ':' \s+ 
        <lat=:.num> ',' <long=:.num> \s+ ':' \s+ 
        <sales=:.integer> \n 
    }
    token name { \w+ }
}
```
Writing the grammar

Finally, some easy tokens to parse numbers

grammar SalesExport {
  token TOP { ^ <country>+ $ }
  token country {
    <name> \n
    <destination>+ 
  }
  token destination {
    \t <name> \s+ ':' \s+
    <lat=.num> ',' <long=.num> \s+ ':' \s+
    <sales=.integer> \n
  }
  token name { \w+ }
  token num { \d+ [\./\d+]? }
  token integer { \d+ }
}
writing the grammar

And we’re done. 😊

grammar SalesExport {
  token TOP { ^ <country>+ $ } 
  token country {
    <name> \n 
    <destination>+ 
  }  
  token destination {
    \t <name> \s+ ':' \s+
    <lat=.num> ',' <long=.num> \s+ ':' \s+
    <sales=.integer> \n
  }
  token name { \w+ }
  token num { \d+ [\./\d+]? } 
  token integer { \d+ } 
}
So we try it out...

$ perl6 travvstats.p6
So we try it out...

$ perl6 travstats.p6

Bool::False
So we try it out...

$ perl6 eg1.p6

Bool::False

Why?
NO
OH NOES
So what can we do?
Give up and go to the pub

Pros
The pub has beer
Beer is good
We can forget about our fail grammar
Give up and go to the pub

**Pros**
The pub has beer
Beer is good
We can forget about our fail grammar

**Cons**
Our grammar is still broken
Give up and go to the pub

**Pros**
The pub has beer
Beer is good
We can forget about our fail grammar

**Cons**
Our grammar is still broken

Dang.
Print statements

We can embed closures in our regexes at pretty much any point we want.

Can use them to work out where we got to, or dump some information for us.
Print statements

We can embed closures in our regexes at pretty much any point we want.

Can use them to work out where we got to, or dump some information for us.

But that’s so 1990s...
Grammar::Tracer

Gives us a trace of all the various rules that our grammar calls as it tries to match

Indicates whether the rule matched or not

When it matches, includes the string that was matched

Tree-like output
Add a use statement for this module, and any grammars in that lexical scope will automatically be traced when invoked

```perl
use Grammar::Tracer;
```

(Live demo!)
Like the tracer, but you can set breakpoints, single step through the grammar, run up until a match failure, run up to a certain rule and so forth...

Again, just add a using statement...

```perl
use Grammar::Debugger;
```

(Live demo!)
About the modules
Pure Perl 6

Implemented using meta-programming and intercepting method dispatch

Grammar::Tracer is about 45 lines

Grammar::Debugger is about 170 lines

On github.com/jnthn later today!
Thank you!
Questions?

Blog: http://6guts.wordpress.com/
Twitter: jnthnwrthngtn
Email: jnthn@jnthn.net

(BTW, my $company is hiring; 😊)