

A Raku Grammar for Navigation Lights

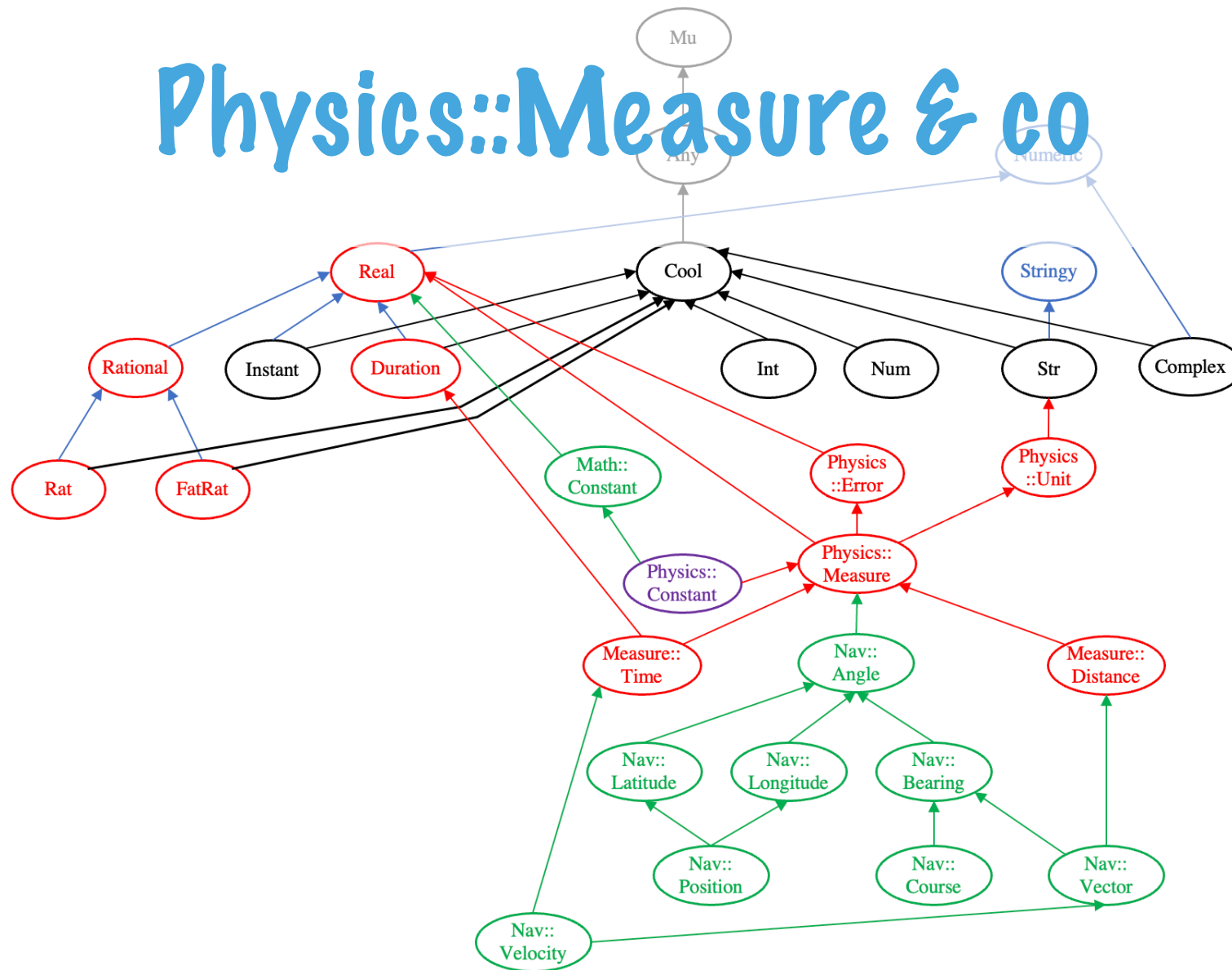
code walkthrough with live examples



One feature that sets raku apart from other programming languages is the built-in Grammar parser syntax. This provides a concise, clean and extensible tool for parsing text and triggering Actions according to the content. Navigation aids such as buoys and markers sport flashing lights with characteristics such as color, duration, phase, occulting, speed, height, visibility and so on are represented on navigation charts by way of a short code e.g. `Fl(4)15s37m28M`.

This talk aims to show how raku provides the average coder (me) with a new practical alternative to Regexp and/or specialist recursive descent modules. It should illustrate how the combination of the raku built-in OO system and Grammars/Actions keeps the problem domain / problem solution in focus via code and visual examples.

Physics::Measure & co



Physics::Navigation

- * Jupyter Notebook Examples....

<https://github.com/p6steve/raku-Yacht-Navigation>

- * To launch with Binder:



- * click the badge above, sometimes the server will be built and takes about 60 secs to launch
- * if you are unlucky, a new server build can take 30-40sec, please be patient (show logs to see the action)

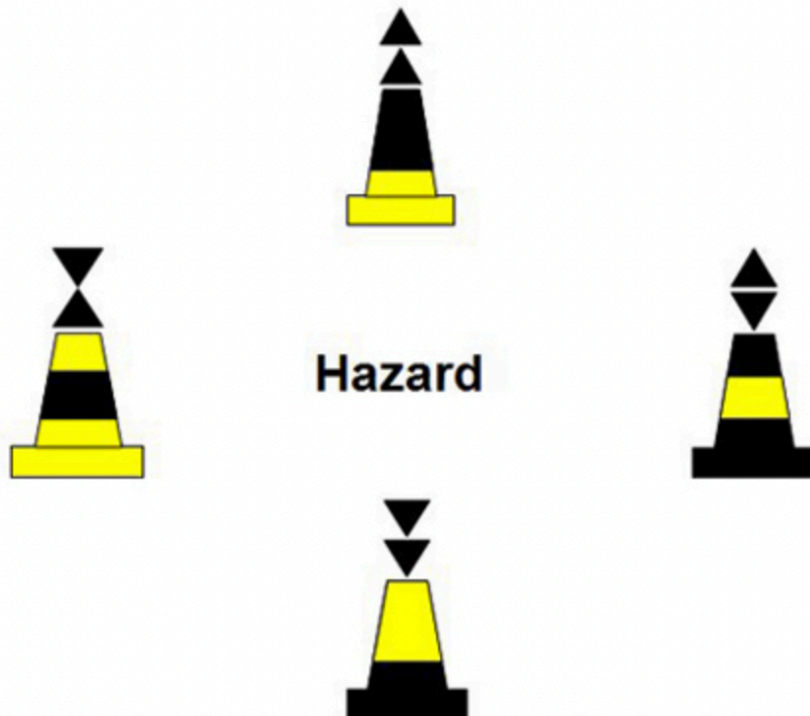
Cardinal Buoys

One interesting set of buoys - often used to indicate hazards - is the Cardinal Buoys.

Four cardinal buoys are defined, North, South, East and West. These can be used to surround a hazard, placed so that:

- North Cardinal Mark is sited to the North of the Hazard - keep to the North of it
- South Cardinal Mark is sited to the South of the Hazard - keep to the South of it
- East Cardinal Mark is sited to the East of the Hazard - keep to the East of it
- West Cardinal Mark is sited to the West of the Hazard - keep to the West of it

Simples!



Define our Buoys

```
[5]: my $ncm = NorthCardinal.new( position => $pos-A ); say "$ncm";
```

NorthCardinal Buoy at (51°30.432'N, 000°7.158'W)
Colours:Black,Yellow. Shapes:Up,Up. Outline:None. Pattern:Layers.
Flashes quickly

```
[6]: my $wcm = WestCardinal.new( position => $pos-B ); say "$wcm";
```

WestCardinal Buoy at (51°30.324'N, 000°7.656'W)
Colours:Yellow,Black,Yellow. Shapes:Down,Up. Outline:None. Pattern:Layers.
Flashes quickly 9 times every 15 seconds

```
[7]: my $ecm = EastCardinal.new( position => $pos-C ); say "$ecm";
```

EastCardinal Buoy at (51°30.324'N, 000°6.66'W)
Colours:Black,Yellow,Black. Shapes:Up,Down. Outline:None. Pattern:Layers.
Flashes quickly 3 times every 10 seconds

```
[8]: my $scm = SouthCardinal.new( position => $pos-D ); say "$scm";
```

SouthCardinal Buoy at (51°30.216'N, 000°7.656'W)
Colours:Yellow,Black. Shapes:Down,Down. Outline:None. Pattern:Layers.
Flashes quickly 6 times plus one long every 15 seconds

A couple of useful mnemonics are:

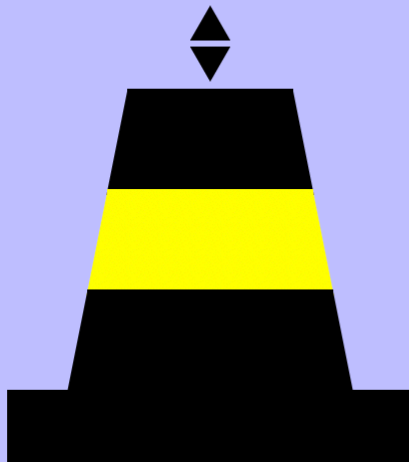
- the ECM top marks look like an "Easter Egg"
- the WCM top marks look like a "W" sideways on

Let's Pick One

```
[11]: my $cm := $ecm;      say $cm.position.Str;  
(51°30.324'N, 000°6.66'W)
```











```
[12]: $cm does BuoyShaped;  
"$cm".say;  
  
my $drawing = Drawing.new( elements => $cm.elements );  
$drawing.serialize.say;
```

EastCardinal+{BuoyShaped} Buoy at (51°30.324'N, 000°6.66'W)
Colours:Black,Yellow,Black. Shapes:Up,Down. Outline:None. Pattern:Layers.
Flashes quickly 3 times every 10 seconds



Light Characteristic

A light characteristic is a graphic and text description of a navigational light sequence or colour displayed on a nautical chart.

Description	Characteristic	Chart Abbreviation
Alternating		Alt. R.W.G.
Fixed		F.
Flashing		Fl.
Group flashing		Gp Fl. (2)
Occulting		Occ.
Group occulting		Gp Occ(3)
Quick flashing		Qk.Fl.
Very quick flashing		V.Qk.Fl.
Isophase		Iso.
Morse		Mo.(letter)


```
my $cm := $scm;  
say $cm.light-svg.^name;  
say $cm.light-svg.Str;
```

```
Physics::Navigation::SVG-animation
duration is 15;
```

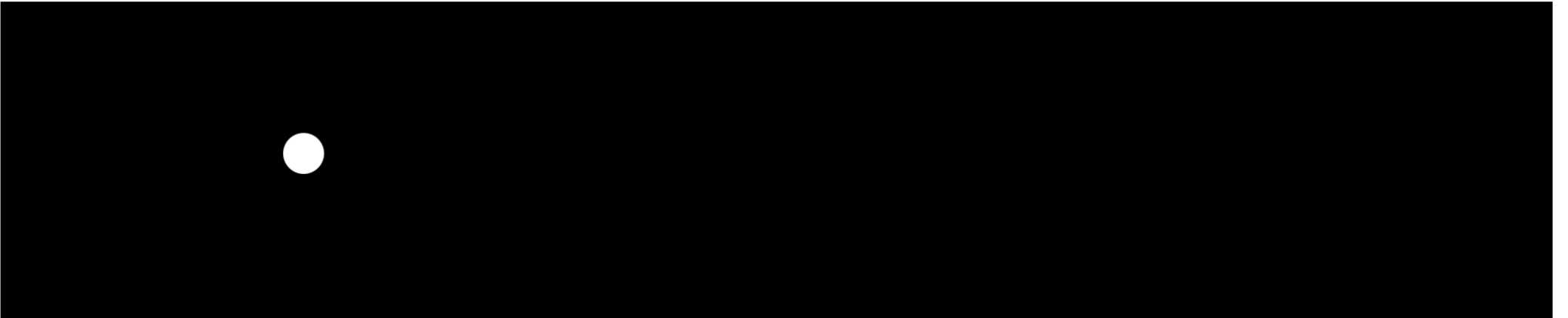
```
pattern is <#fff #000 #fff #000 #fff #000 #fff #000 #fff #000 #fff #000 #000 #000 #fff #fff #fff #fff #fff #fff #000 #000 #000 #000 #000 #000 #000 #000 #000 #000 #000 #000>;
```

... and Draw It

```
$cm does LightShaped;
"$cm".say;
```

```
my $drawing = Drawing.new( elements => $cm.elements );
$drawing.serialize.say;
```

SouthCardinal+{LightShaped} Buoy at (51°30.324'N, 000°6.66'W)
Colours:Yellow,Black. Shapes:Down,Down. Outline:None. Pattern:Layers.
Flashes quickly 6 times plus one long every 15 seconds




```
##### Buoys and Lights #####
```

```
enum IALA      is export <A B>;                                #EMEA, Americas
enum Pattern   is export <Solid Layers Stripty>;
enum Outline   is export <Can Cone None>;
enum Shape     is export <Ball Cross Down Up>;                #[0] is top of Buoy
enum Colour    is export <Black Green Red White Yellow>;      #[0] is top of Buoy
```

```
our $IALA is export;
our %iala-colours = %( A => [Red, Green], B => [Green, Red] ); #B => red right returning
```

```
#for example, this code FL(4)15s37m28M should produce this sentence...
#"Flashes 4 times every 15 seconds at height of 37m above MHWS range 28Miles in clear visibility."
```

```
#use Grammar::Tracer;      #uncomment for debug
```

```
grammar LightCode {
  token TOP      { <kind> ['.']? <group>? <colour>? <extra>? <period>? <height>? <visibility>? }

  token kind      { <veryquick> | <quick> | <flashing> | <fixed> | <isophase> | <occulting> }
  token veryquick { 'VQ' }
  token quick     { 'Q' }
  token flashing  { 'FL' }
  token fixed     { 'F' }
  token isophase  { 'Iso' }
  token occulting { 'Oc' }

  token group     { '(' <digits> ')' }
  token colour    { <[GRW]>+ }
  token extra     { '+L FL.' }
  token period    { <digits> 's' }
  token height    { <digits> 'm' }
  token visibility { <digits> 'M' }
  token digits    { \d+ }
```



```

class LightCode-actions {
  method TOP($/) {
    my @p;
    @p.push: $/ <kind>.made;
    @p.push: $/ <group>.made;
    @p.push: $/ <colour>.made;
    @p.push: $/ <extra>.made;
    @p.push: $/ <period>.made;
    @p.push: $/ <height>.made;
    @p.push: $/ <visibility>.made;

    $/.make: @p.grep({.so}).join(' ')
  }
  method kind($/) {
    given $/ {
      when 'VQ' { $/.make: 'Flashes very quickly' }
      when 'Q' { $/.make: 'Flashes quickly' }
      when 'FL' { $/.make: 'Flashes' }
      when 'F' { $/.make: 'Fixed' }
      when 'Iso' { $/.make: 'Isophase' }
      when 'Oc' { $/.make: 'Occulting' }
    }
  }
  method group($/) {
    $/.make: ~$/ <digits> ~ ' times'
  }
  method colour($/) {
    my %palette = %( G => 'green', R => 'red', W => 'white' );
    $/.make: %palette{~$/{
  }
  method extra($/) {
    $/.make: 'plus one long'
  }
  method period($/) {
    $/.make: 'every ' ~ ~$/ <digits> ~ ' seconds'
  }
  method height($/) {
    $/.make: 'at height of ' ~ ~$/ <digits> ~ 'm above MHWS'
  }
  method visibility($/) {
    $/.make: 'range ' ~ ~$/ <digits> ~ 'nmiles in clear visibility'
  }
}

```




```

role Light is export {
  has Str $.light-defn = '';

  method light( --> Str ) {
    LightCode.parse($.light-defn, actions => LightCode-actions.new).made
  }

  method light-svg {
    LightCode.parse($.light-defn, actions => LightCodeSVG-actions.new).made
  }
}

class Buoy does Light is export {
  has Position $.position is required;

  has Pattern $.pattern = Layers;
  has Outline $.outline = None;
  has Colour @.colours = [];
  has Shape @.shapes = [];

  method Str( --> Str ) {
    my $name = self.^name;
    $name ~~ s/'Physics::Navigation::'//;
    qq:to/END/;
    $name Buoy at {self.position}
    Colours:{@.colours.join(',')}. Shapes:{@.shapes.join(',')}. Outline:{$.outline}. Pattern:{$.pattern}.
    {self.light}
    END
  }
}

class Lateral is Buoy is export {
  has Pattern $.pattern = Solid;

  multi method colours( Int $i --> Array ) {
    [%iala-colours{$IALA}[$i],]
  }

  method light-defn( --> Str ) {
    my $ci = $.colours[0].substr(0,1).uc;
    "FL.{$ci}5s"
  }
}

class PortLateral is Lateral is export {
  has Outline $.outline = Can;
  multi method colours { samewith( 0 ) }
}

```

```

#| desired output is SVG-animation object
class SVG-animation is export {
  has $.base-rate = 1;
  has $.continuous is rw = False;
  has $.duration is rw = 5;          #total duration/period (s) of flash sequence
  has $.fl-times is rw = 1;          #number of times to flash
  has $.on is rw = '#fff';
  has $.off is rw = '#000';
  has $.extra is rw = False;
  has $.special = '';

  # @.pattern = <#800 #f00 #800 #800>;
  #          ^^^^ - colour codes (#RGB)
  #          - alternate on/off
  #          - .elems determined by (=duration/base-rate)

  method pattern {
    my $beats = $!duration / ( $!base-rate * 2 ); #issue 2 phases per beat

    given $!special {
      when 'Fixed' {
        return( $!on xx ( 2 * $beats ) )
      }
      when 'Isophase' {
        return( $!on xx $beats, $!off xx $beats )
      }
      when 'Occulting' {
        ($!on, $!off) = ($!off, $!on)
      }
    }

    my @p;
    for ^$beats {
      if $!continuous || $!fl-times >= 1 {
        @p.push: $!on, $!off;
      } else {
        @p.push: $!off, $!off;
      }
      $!fl-times -= 1;          # -- does not work on attributes
    }

    if $!extra {
      my $ex-start = $beats;          # extra = half and half
      my $ex-beats = 3 / $!base-rate; # long = 3s

      my @e;
      for ^$ex-beats {
        @e.push: $!on;
      }
      @p.splice( $ex-start.Int, $ex-beats.Int, @e );
    }
    @p
  }

  method Str {
    "duration is {$.duration};\n pattern is <{$.pattern}>;\n\n";
  }
}

```



```

class LightCodeSVG-actions {
  method TOP($/) {
    #dd $<kind>;

    my $anime = $<kind>.made;

    with $<period>.made {$anime.duration = $_};
    with $<group>.made {$anime.fl-times = $_; $anime.continuous = False };
    with $<colour>.made {$anime.on = $_};
    with $<extra>.made {$anime.extra = $_};

    $/.make: $anime;

    #put "in TOP..."; dd $/.made;
  }
  method kind($/) {
    # Flashes = 1s, Quick = 1/2s, V. Quick = 1/4s
    my ( $base-rate, $continuous, $special );

    given $/ {
      when 'VQ' { $base-rate = <1/4>; $continuous = True }
      when 'Q' { $base-rate = <1/2>; $continuous = True }
      when 'FL' { $base-rate = 1 }
      when 'F' { $special = 'Fixed' }
      when 'Iso' { $special = 'Isophase' }
      when 'Oc' { $special = 'Occulting' }
    }
    $/.make: SVG-animation.new( :$base-rate, :$continuous, :$special );
  }
  method group($/) {
    $/.make: ~$<digits>.Int
  }
  method colour($/) {
    my %palette = %( G => '#0f0', R => '#f00', W => '#fff' );
    $/.make: %palette{~$/>
  }
  method extra($/) {
    $/.make: True
  }
  method period($/) {
    $/.make: $<digits>.Int
  }
}

```

Raku wordy style

```
my $number = prompt('Enter number> ');

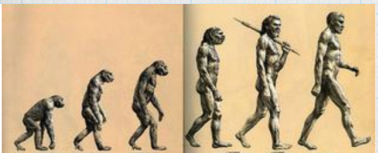
$number
    .comb
    .unique
    .sort
    .join(', ')
    .say;
```

<https://andrewshitov.com/2019/09/09/unique-digits-in-perl-6/>

Raku regex style (i)

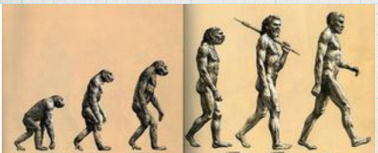
```
my regex float { <[+-]>?\d* ' . ' \d+[e<[+-]>?\d+]? }
```

<https://docs.raku.org/language/regexes-best-practices>




Raku regex style (ii)

```
my regex float {  
    <[+-]>?      # optional sign  
    \d*         # leading digits, optional  
    '.'         # decimal point  
    \d+         # digits after decimal  
    [          # optional exponent  
        e <[+-]>? \d+  
    ]?  
}
```

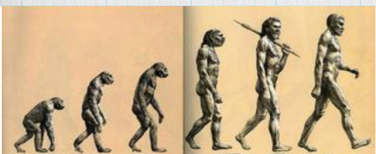


<https://docs.raku.org/language/regexes-best-practices>

p6steve.com 

Raku regex style (iii)

```
my token sign { <[+-]> }  
my token decimal { \d+ }  
my token exponent { 'e' <sign>? <decimal> }  
my regex float {  
    <sign>?  
    <decimal>?  
    '.'  
    <decimal>  
    <exponent>?  
}
```



<https://docs.raku.org/language/regexes-best-practices>

p6steve.com 

Thanks! Questions?



Useful Links

- * <https://github.com/bduggan/p6-jupyter-kernel>
- * <https://docs.raku.org/language/grammars>
- * https://en.wikipedia.org/wiki/Light_characteristic
- * <https://github.com/p6steve/raku-Physics-Navigation>
- * <https://github.com/p6steve/raku-Physics-Measure>

