Perl6?
Rakudo?
6Lang?

A Primer

Who?

Why?

Unicode

```
my $á1 = "\c[LATIN SMALL LETTER A WITH ACUTE]";

my $á2 = "a\x301";

say "$á1 : $á2";
á : á

say $á1 eq á2;
True

say "$á1 : $á2".uc;
Á : Á

my $æ = 2;
$æ = ( $æ * ¾ )²;
say "æ => $æ";
æ => 2.25
```

High level concurrent and asynchronous methods

- Promises
- Supplies (Event Streams)
- Channels (Thread Safe FIFO Data Queues)
- Proc::Async (Handle external processes in an async fashion)
- Junctions (Threaded superposition of possible values)
- Parallel Iterables (ordered and unordered results)

Also a complete low level threading model for those who need it.

Powerful OO system and typing

Moose-like with Roles and Meta Objects

Functions and Signatures are Objects (**everything** is an object)

Progressive typing system, method signatures and multi-methods

Use as much or as little of it as you want.

Other Stuff

- Runs on MoarVM and on JVM with a few known bugs (being fixed)
- Sets with complete Set Theory operators
- Lazy lists, sequences and iterables
- Grammars allowing for even more complex data parsing
- NativeCall allows for simple integration with C and (on JVM) Java libraries
  - (No XS required)
- Inline::Perl lets you use Perl5 code inside your Perl6 code
  - If you've compiled Perl5 to do it that is.
- Rational Numbers by default
  - (0.1 + 0.2 - 0.3 == 0 !)
- And much much more.

Crazy Example

```
subset UKPostCode of Str where * ~~ rx:i/^[
                                        <[A..PR..UWYZ0..9]>
                                        <[A..HK..Y0..9]>
                                        <[AEHMNPRTVXY0..9]>?
                                        <[ABEHMNPRVWXY0..9]>?
                                        ' ' ** 1..2
                                        <[0..9]>
                                        <[ABD..HJLN..UW..Z]> ** 2
                                        |GIR ' ' 0AA$
                                        ]$/;

multi checkPC( UKPostCode $pc ) { True }
multi checkPC( $_ ) { False }

say checkPC( "se1 2lh" );
True

say checkPC( "BOBS HOUSE" );
False
```

Another Crazy Example

```
module SI {
    role Unit {
        has Numeric $.size;
        has Str $.descriptor;

        method Str(--> Str) {
            return "{$.size}{$.descriptor}";
        }
    }
    class Distance does SI::Unit {
        submethod BUILD( Numeric :$!size ) { $!descriptor = 'm';
}
    }
    sub postfix:<m>( Numeric $size --> SI::Distance ) is export {
        return SI::Distance.new( size => $size );
    }
    class Area does SI::Unit {
        submethod BUILD( Numeric $!size ) { $!descriptor = 'm²';
}
    }
    multi postfix:<m²>( Numeric $size --> SI::Area ) is export {
        return SI::Area.new( size => $size );
    }
    multi postfix:<²>( SI::Distance $length --> SI::Area )
        is export {
        my $l_n = $length.size;
        return SI::Area.new( size => $l_n * $l_n );
    }
    multi infix:<*>(
        SI::Distance $height, SI::Distance $width --> SI::Area
    ) is export {
        my $h_n = $height.size;
        my $w_n = $width.size;
        return SI::Area.new( size => $h_n * $w_n );
    }
}
```

Still working on this one but it's a simple system to let you write SI unit based equations in code.

```
my $h = 10m;
my $w = 20m;
my $a = $h * $w;

say $a; 200m2
```