

Rakudo 楽土

Polyglot Programming DC 2014

Brock Wilcox

awwaiid@thelackthereof.org

bwilcox@optoro.com

Big language. Lots of stuff.

Object Oriented (Ruby, CLOS)

Data/Function Oriented (Haskell, Clojure)

Operator Oriented (APL, J)

Sigil Oriented (Ruby, Perl)

Optional Static Typing (Common Lisp)

Multi Dispatch (Clojure, Haskell)

Normal Stuff

Fancy Stuff

Insane Stuff

Normal Stuff

- Garbage collected
- Curley, semi-colon
- Class object system
- Roles (like interfaces, mixins, traits)
- Scalars, lists, hashes, sets
- Block scoping, closures, anon funcs

```
class Animal { }
```

```
class Dog is Animal { }
```

```
role Logging { }
```

```
class Dog does Logging { }
```



```
class Person {  
  has $.name;  
  has $.age;  
  
  method say-hi {  
    say "I am the great $.name! I am $.age years old.";  
  }  
}
```

```
my $joe = Person.new( name => 'Joe', age => 37 );
```

```
$joe.say-hi
```

Sigils / Twigls

```
$joe          # scalar
$!name       # private instance var
$.name       # public instance var kinda
@people      # list
%phonebook   # hash
&lookup      # callable block
```

Scalars, Lists, Hashes

```
my @names = ('Casey', 'Dakota', 'Jaiden', 'Jordan', 'Peyton');  
my @names = <Casey Dakota Jaiden Jordan Peyton>;  
say "Third: @names[2]"  
say @names.join(", ");
```

```
my %ages = {  
  Casey => 5,  
  Dakota => 10,  
  Jaiden => 15,  
};
```

```
say "Jaiden is %ages{'Jaiden'}";  
say "Jaiden is %ages<Jaiden>";
```

Ruby-style DSL blocks

```
sub doit(&thing) {  
    say "I say...";  
    &thing();  
}
```

```
doit { say "hello" }
```


Closures / Lambdas

```
sub counter {  
    my $n = 1;  
    -> { $n++ };  
}
```

```
my &counter_1 = counter();  
my &counter_2 = counter();
```

```
say &counter_1(); # 1  
say &counter_1(); # 2  
say &counter_2(); # 1  
say &counter_2(); # 2
```

Fancy Stuff

- Optional Static Typing
- Introspection and MOP
- Advanced subroutine argument declarations
- Multi dispatch (both type and value based)
- Generators
- Lazy evaluated lists
- Partial application / currying
- Concurrent multi-version module usage

```
my $x = "fishies"
```

```
my Int $x = "fishies" # ERROR
```

```
sub add_only_ints(Int $x, Int $y) {  
    $x + $y  
}
```

Multi-dispatch
(pattern matching)

```
multi sub add_stuff(Int $x, Int $y) {  
    $x + $y  
}
```

```
multi sub add_stuff(Str $x, Str $y) {  
    $x ~ $y  
}
```


Meta Programming / Introspection

```
# Get the class  
say 3.WHAT # (Int)
```

```
# Get the heirarchy  
3.^mro
```

```
# Get the methods  
3.^methods
```

Insane(ly awesome) Stuff

- Operator overloading
- Meta/Hyper operators
- Chained comparisons
- Adverbs
- Grammars
- Junction values
- Unixy MAIN
- Macros
- Whatever-star
- Placeholder variables

Operator-Oriented

$$5 + 7$$

Meta/Hyper Operators

$$\$X + = 5$$

$$\$X - = 5$$

$$\$X * = 5$$

$$\$X / = 5$$

my \$x = 5

\$x = \$x.is_prime

\$x .= is_prime

[+] <5 7 23 21 32>
88

<1 2 3 4> <<+>> <5 6 7 8>
6 8 10 12

<1 2 3 4> «+» <5 6 7 8>
6 8 10 12

<1 2 3 4> «*» <5 6 7 8>
5 12 21 32

User-defined operators

```
sub infix:<◇> ($a, $b) {  
  $a >= $b ?? $a !! $b;  
  # or: $a max $b  
}
```

17 ◇ 42

`my $x = 7;`

`$x ◇ = 3;`

```
sub postfix:<!!>($n) {  
  [*] 2..$n;  
}
```

6! # 720

"I swear the only reason we don't have factorial as a standard operator in the language, is so that we can impress people by defining it."

- Carl Mäsak

type =====	position =====	syntax =====
prefix	before a term	!X
infix	between two terms	X ! Y
postfix	after a term	X!
circumfix	around	[X]
postcircumfix	after & around	X[Y]

Get a list of all builtin infix operators

```
CORE::.keys.grep(/infix/)>>.say
```

Show all the multi dispatches for '+'

```
&[+] .candidates>>.say
```

Random Stuff

$$2 < x < 10$$

`$x = 5|7`

`if $x == 5 { say "yep" } else { say "nope" }`

```
$x = 5&7
```

```
if $x.is-prime { say "yep" } else { say "nope" }
```


@stuff.map: { $\$^a + 2$ }

@stuff.map: { $\$^{fish} + \sticks }

$$\rightarrow \{ x + 2 \}$$

<2 3 4 5 4> <<+>> 2

<2 3 4 5 4>.map: -> \$x { \$x + 2 }

<2 3 4 5 4>.map: * + 2

```
use MONKEY_TYPING;
```

```
augment class Int {  
  method infix: <◇> ($v) {  
    self min $v  
  }  
}
```

In-progress features:

- Non-blocking IO
- Inline concurrency
- Autothreading
- Advanced macros
- Improving JVM integration

THE END

Oh yeah. Almost forgot.

Rakudo is an implementation of Perl6