Perl 6 Signatures: The Full Story

Jonathan Worthington

Some Basic Examples

Some Basic Examples

(From Perl 5 => Perl 6)

Positional Parameters

```
sub get_coordinates {
    my ($city, $country) = @_;
    …
}
```

```
sub get_coordinates($city, $country) {
    …
}
```

Named Parameters

```
sub get_capital {
    my %params = @_;
    my $country = $params{'country'};
    …
}
```

```
sub get_capital(:$country) {
    …
}
```

Slurpy Positionals

```
sub sort_west_to_east {
    return sort {
        $a->latitude <=> $b->latitude
    }, @_;
}
```

```
sub sort_west_to_east(*@places) {
    return @places.sort({
        $^a.latitude <=> $^b.latitude
    });
}
```

By the way…

```
sub sort_west_to_east(*@places) {
    return @places.sort({
        $^a.latitude <=> $^b.latitude
    });
}
```

…can also be written in Perl 6 as…

```
sub sort_west_to_east(*@places) {
    return @places.sort({ .latitude });
}
```

(like sorting on the mapped values)

And even prettier…

```
sub sort_west_to_east(*@places) {
    return @places.sort({ .latitude });
}
```

…can also be written in Perl 6 as…

```
sub sort_west_to_east(*@places) {
    return @places.sort(*.latitude);
}
```

(because `*.foo` generates a closure like `{ $_.foo }`)

Slurpy Nameds

```
sub sum_distances {
    my %place_distances = @_;
    my $total = 0;
    $total += $_ for values %place_distances;
    return $total;
}
```

```
sub sum_distances(*%place_distances) {
    return [+] %place_distances.values
}
```

Arity Checking

The Perl 6 runtime checks that you passed enough parameters.

If you pass too few or too many, an exception is thrown.

```
sub book_train($from, $to, $date, $time) {
    …
}
book_train('Kiev', 'Lviv', '2010-6-13');
```

```
Not enough positional parameters passed; got 3 but expected 4
  in 'book_train' at line 1
  in main program body at line 4
```

Optional Parameters

```
sub book_train {
    my ($from, $to, $date, $time) = @_;
    …
}
```

```
sub book_train($from, $to, $date, $time?) {
    …
}
```

Defaults

```
sub biggest_city {
    my $country = shift;
    my $rank    = shift || 1;
    …
}
```

```
sub biggest_city($country, $rank = 1) {
    …
}
```

Required Named Parameters

While positional parameters are required by default, named parameters are optional by default.

To require one be passed, use !

```
sub book_train(:$from!, :$to!,
               :$date!, :$time) {
    …
}
```

Parameter Binding

Perl 6 Signatures: The Full Story

In Perl 5, you get a copy of the arguments to work with in @_.

In Perl 6, parameters are bound. This means that you get a (by defualt) read-only alias to the original value.

Read-only Alias

In Perl 6, this code will fail:

```
sub convert_currency($amount, $rate) {
    $amount = $amount * $rate;
    return $amount;
}

my $price = 99;
$price = convert_currency($price, 11.1);
say $price;
```

```
Cannot assign to readonly value
  in 'convert_currency' at line 2:test.p6
  in main program body at line 6:test.p6
```

is copy

To make this work like in Perl 5, explicitly indicate we want a copy

```
sub convert_currency($amount is copy, $rate) {
    $amount = $amount * $rate;
    return $amount;
}

my $price = 99;
$price = convert_currency($price, 11.1);
say $price;
```

```
1098.9
```

is rw

Can also modify the original without having to pass a reference

```
sub convert_currency($amount is rw, $rate) {
    $amount = $amount * $rate;
}

my $price = 99;
convert_currency($price, 11.1);
say $price;
```

```
1098.9
```

Passing Arrays / Hashes

In Perl 6, passing an array or hash works like passing a reference.

```
sub example(@array, %hash) {
    say @array.elems;
    say %hash.keys.join(', ');
}

my @numbers = 1,2,3,4;
my %ages = Jnthn => 25, Noah => 120;
example(@numbers, %ages);
```

```
4
Noah, Jnthn
```

Types

What are types?

In Perl 6, every value knows its type.

```
say 42.WHAT;
say "camel".WHAT;
say [1, 2, 3].WHAT;
say (sub ($n) { $n * 2 }).WHAT;
```

```
Int()
Str()
Array()
Sub()
```

A type name in Perl 6 represents all possible values of that type.

Type Constraints

Can restrict a parameter to only accept arguments of a certain type.

```
sub show_dist(Str $from, Str $to, Int $kms) {
    say "From $from to $to is $kms km.";
}
show_dist('Kiev', 'Lviv', 469);
show_dist(469, 'Kiev', 'Lviv');
```

```
From Kiev to Lviv is 469 km.
Nominal type check failed for parameter '$from'; expected Str
but got Int instead
  in 'show_dist' at line 1:test.p6
  in main program body at line 5:test.p6
```

Type Coercions

Sometimes, you want to accept any type, but then transform it into another type before binding to the parameter

```
sub show_dist($from, $to, $kms as Int) {
    say "From $from to $to is $kms km.";
}
show_dist('Kiev', 'Lviv', '469');
show_dist('Kiev', 'Lviv', 469.35);
```

```
From Kiev to Lviv is 469 km.
From Kiev to Lviv is 469 km.
```

Constraints

Sometimes, you need to do some more powerful validation on arguments.

```
sub discount($price, $percent
                where (1 <= $percent <= 100)) {
    say "You get $percent% off! Pay EUR " ~
        $price - ($price * $percent / 100);
}
discount(100, 20);
discount(100, 200);
```

```
You get 20% off! Pay EUR 80
Constraint type check failed for parameter '$percent'
  in 'discount' at line 2:test.p6
  in main program body at line 7:test.p6
```

Warning!

Be careful about using type constraints on arrays and hashes. The type constraints the elements.

```
sub total(Array @distances) {
     # WRONG! Takes an Array of Arrays!
}
```

```
sub total(Int @distances) {
     # Correct, takes an array of Ints.
}
```

Multiple Dispatch

Perl 6 Signatures: The Full Story

In Perl 6, you can write many subs with the same name but different signatures.

When you call the sub, the runtime will look at the types of the arguments and pick the best match.

Dispatch By Arity

Example (from Test.pm): dispatch by different number of parameters

```
multi sub todo($reason, $count) is export {
    $todo_upto_test_num = $num_of_tests_run + $count;
    $todo_reason = '# TODO ' ~ $reason;
}

multi sub todo($reason) is export {
    $todo_upto_test_num = $num_of_tests_run + 1;
    $todo_reason = '# TODO ' ~ $reason;
}
```

Dispatch By Type

Example: part of a JSON emitter

```
multi to-json(Array $a) {
    return '[ ' ~
        $a.values.map({ to-json($_) }).join(', ') ~
        ' ]';
}
multi to-json(Hash $h) {
    return '{ ' ~
        $h.pairs.map({
            to-json(.key) ~ ': ' ~ to-json(.value)
        }).join(', ') ~
        ' }';
}
```

Dispatch By Constraint

Can use multiple dispatch with constraints to do a lot of "write what you know" style solutions

Factorial:

Factorial:

fact(0) = 1

Factorial:

fact(0) = 1

fact(n) = n * fact(n - 1)

Factorial:

fact(0) = 1

fact(n) = n * fact(n - 1)

```
multi fact(0)  { 1 }
multi fact($n) { $n * fact($n - 1) }
```

Factorial:

fact(0) = 1

fact(n) = n * fact(n - 1)

```
multi fact(0)  { 1 }
multi fact($n) { $n * fact($n - 1) }
```

```
(Int $ where 0)
```

Fibonacci Sequence:

$$fib(0) = 0$$

$$fib(1) = 1$$

$$fib(n) = fib(n - 1) + fib(n - 2)$$

Fibonacci Sequence:

$$fib(0) = 0$$

$$fib(1) = 1$$

$$fib(n) = fib(n - 1) + fib(n - 2)$$

```
multi fib(0)  { 0 }
multi fib(1)  { 1 }
multi fib($n) { fib($n - 1) + fib($n - 2) }
```

Nested Signatures

Captures

A set of parameters form a signature.

A set of arguments from a capture.

Signature

```
sub greet($name, :$greeting = 'Hi') {
    say "$greeting, $name!";
}
greet('Лена', greeting => 'Привет');
```

Capture

Coercing To Captures

It is possible to coerce arrays, hashes and other objects into captures.

Array elements => positional arguments

Hash pairs => named arguments

Object attributes => named arguments

Unpacking Arrays

Can extract elements from within an array, to do FP-style list processing

```
sub head([$head, *@tail]) {
    return $head;
}
sub tail([$head, *@tail]) {
    return @tail;
}
my @example = 1,2,3,4;
say head(@example);
say tail(@example);
```

```
1
234
```

Unpacking Hashes

Can extract values by key

```
sub show_place((:$name, :$lat, :$long, *%rest)) {
    say "$name lies at $lat,$long.";
    say "Other facts:";
    for %rest.kv -> $title, $data {
        say "    $title.ucfirst(): $data";
    }
}
my %info = name => 'Kiev', lat => 50.45,
           long => 30.52, population => 2611300;
show_place(%info);
```

```
Kiev lies at 50.45,30.52.
Other facts:
    Population: 2611300
```

Unpacking Objects

Can extract values by attribute (only those that are declared with accessors)

```
sub nd($r as Rat (:$numerator, :$denominator)) {
    say "$r = $numerator/$denominator";
}
nd(4.2);
nd(3/9);
```

```
4.2 = 21/5
0.333333333333333 = 1/3
```

Unpacking + Multiple Dispatch

When using multiple dispatch, whether we can unpack a parameter or not works like a constraint.

Therefore we can do multiple dispatch based upon the shape and values inside of complex data structures.

Example: Quicksort

Example: Quicksort

```
# Empty list sorts to the empty list
multi quicksort([]) { () }
```

Example: Quicksort

```
# Empty list sorts to the empty list
multi quicksort([]) { () }

# Otherwise, extract first item as pivot...
multi quicksort([$pivot, *@rest]) {
    ...
}
```

Example: Quicksort

```
# Empty list sorts to the empty list
multi quicksort([]) { () }

# Otherwise, extract first item as pivot...
multi quicksort([$pivot, *@rest]) {
    # Partition.
    my @before = @rest.grep(* < $pivot);
    my @after  = @rest.grep(* >= $pivot);
    …
}
```

Example: Quicksort

```
# Empty list sorts to the empty list
multi quicksort([]) { () }

# Otherwise, extract first item as pivot...
multi quicksort([$pivot, *@rest]) {
    # Partition.
    my @before = @rest.grep(* < $pivot);
    my @after  = @rest.grep(* >= $pivot);

    # Sort the partitions.
    (quicksort(@before), $pivot, quicksort(@after))
}
```

Conclusions

Not Just Replacing @_

Perl 6 signatures provide you with a neater way to handle arguments passed to subs and methods than working with @_.

However, they offer a lot more power, and have applications beyond where you would traditionally use a signature.

"When?"

All of the examples shown today are already working in Rakudo Perl 6.

Signature handling and multiple dispatch are amongst the most mature and stable parts of Rakudo.

Спасибо

Questions?