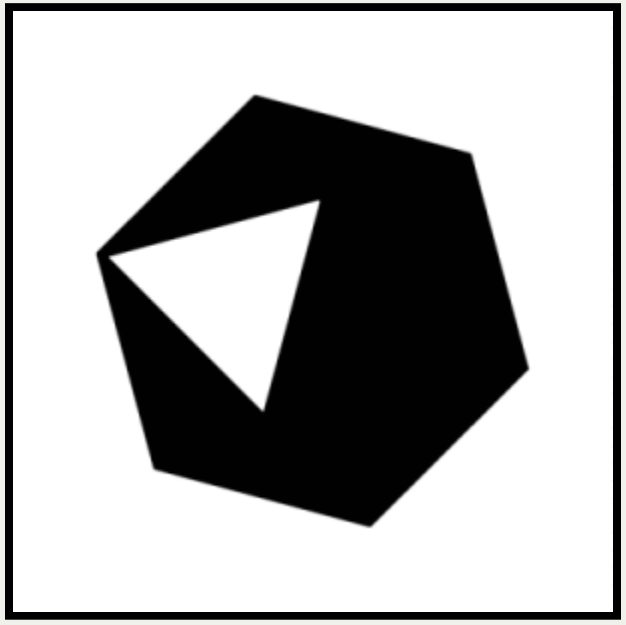
Crystal - better Ruby?



Radosław Bułat

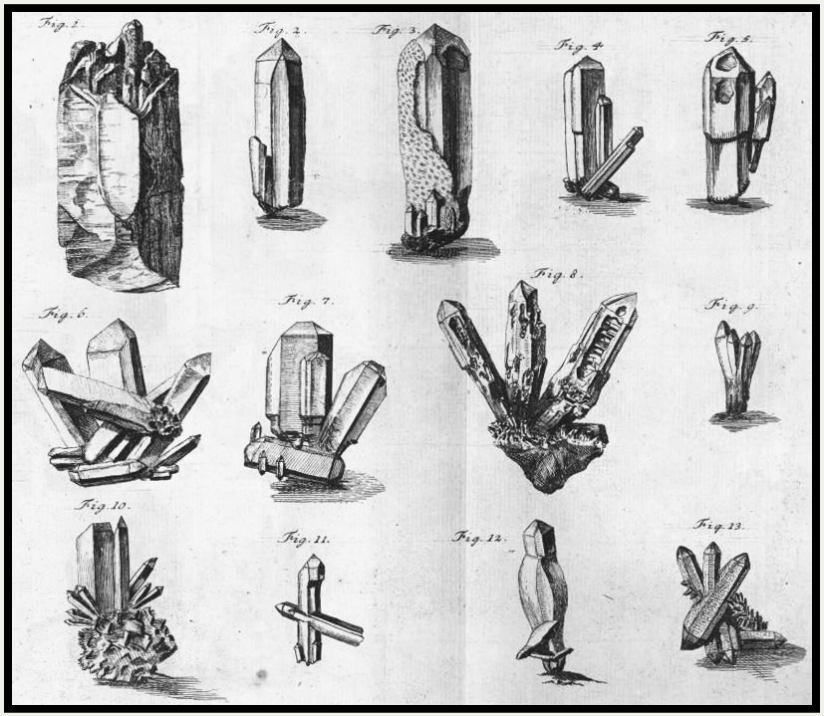
KRUG, grudzień 2016r.

About me

I work at Codest.

We develop webapps but sometimes we make things just for fun.

What is Crystal?



Crystal is a programming language with following characteristics

- syntax very similar to Ruby
- statically type-checked
- have compile-time evaluation and generation of code
- compile to efficient native code

How much Crystal is similar to Ruby?

Syntax

```
puts "Hello World!"

arr = [1, 2, 3, 4, 5]
puts "#{arr} includes 3" if arr.includes?(3)

if m = "--123--".match(/(\d+)/)
  puts m[0].to_i
end

puts "127.0.0.1".split(".").reverse.join(".")
```

OOP - classes

```
class Greeter
  def initialize(name = "there")
    @name = name
  end

  def salute
    puts "Hello #{@name}!"
  end
end

Greeter.new("world").salute
Greeter.new.salute
```

OOP - mixins

```
module Mixin
  def foo
    puts "foo from Mixin called"
  end
end

class MyClass
  include Mixin
end

MyClass.new.foo
```

Blocks & Procs

```
def twice
  yield "here"
  yield "there"
end

twice { |s| puts "Hello #{s}!" }

f = -> { puts "Proc called" }
twice(&f)
```

API

```
puts 1.upto(999)
  .select { |e| e % 3 == 0 || e % 5 == 0 }
  .reduce { |acc, e| acc + e }
```

```
File.open("/etc/hosts") do |file|
  file.each_line { |line| puts line }
end
```

Core classes & modules

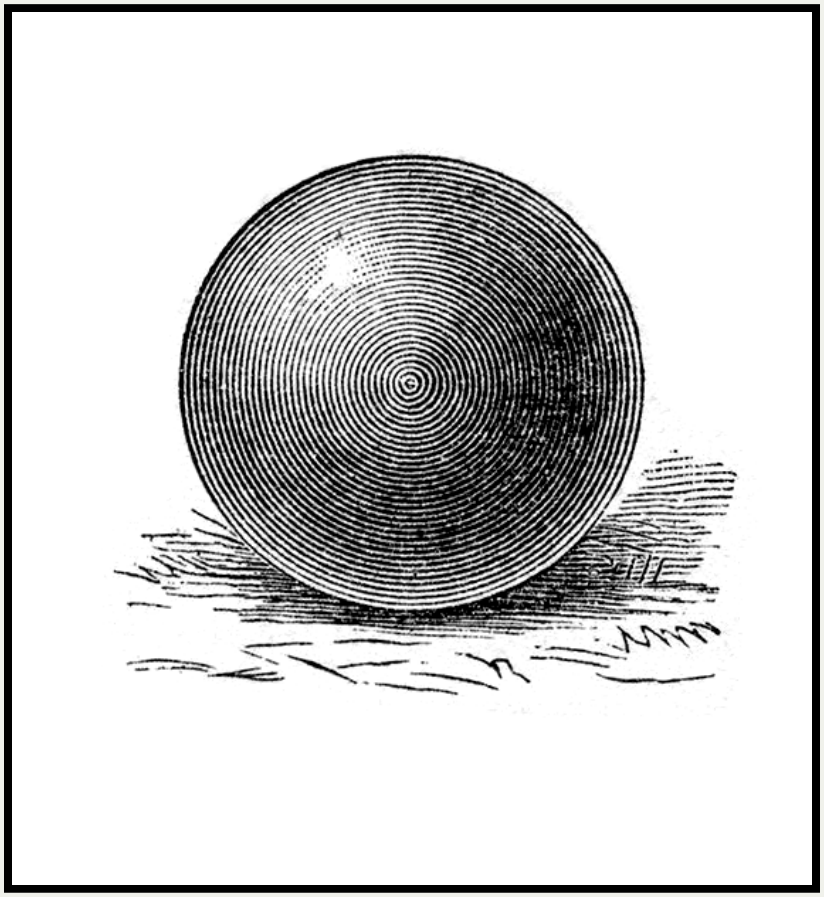
Array, String, Class, Enumerable, Comparable, ENV, File, Hash, Nil, Range, Regex...

Looks familiar?

Learning curve for Ruby programmers is very gentle.

You can use the same programming techniques, best practices and idioms.

Crystal is statically typed



Specifying types in most cases is optional.

Compiler still can verify types during compilation phase.

Type inference

```
a = 1         # Int32
puts a + 123 # Int32#+(Int32) exists
```

```
$ crystal build type_inference1.cr
$ ./type_inference1
124
```

```
a = 1           # Int32
puts a + "123" # Int32#+(String) - doesn't exist
```

```
$ crystal build type_inference2.cr
Error in type_inference2.cr:2: no overload matches 'Int32#+' with type String
Overloads are:
 - Int32#+(other : Int8)
 - Int32#+(other : Int16)
 - Int32#+(other : Int32)
 - Int32#+(other : Int64)
 - Int32#+(other : UInt8)
 - Int32#+(other : UInt16)
 - Int32#+(other : UInt32)
 - Int32#+(other : UInt64)
 - Int32#+(other : Float32)
 - Int32#+(other : Float64)
 - Number#+()

puts a + "123"
       ^
```

Can variable store multiple types?

```
a = (rand(2) == 0 ? 1 : "1") # Int32 | String
puts a * 2                   # both Int32#*(Int32)
                             # and String#*(Int32) exist
```

```
$ crystal run type_inference3.cr
2
$ crystal run type_inference3.cr
11
```

```
a = (rand(2) == 0 ? 1 : "1") # Int32 | String
puts a.abs                   # String#abs doesn't exist
```

```
$ crystal build type_inference4.cr
Error in type_inference4.cr:2: undefined method 'abs' for String
(compile-time type is (Int32 | String))

puts a.abs
       ^~~
```

```
a = (rand(2) == 0 ? 1 : "1") # Int32 | String
if a.is_a?(Int32)
  puts a.abs                 # Int32#abs
end
```

What about method arguments?

```
def add(a : Int32, b : Int32)
  a + b
end

puts add(10, 20)
```

```
30
foobar
```

```
def add(a : Int32, b : Int32)
  a + b
end

puts add("foo", "bar")
```

```
Error in type_inference7.cr:5: no overload matches 'add' with types String, String
Overloads are:
 - add(a : Int32, b : Int32)

puts add("foo", "bar")
     ^~~
```

Type inference applies to method arguments too.

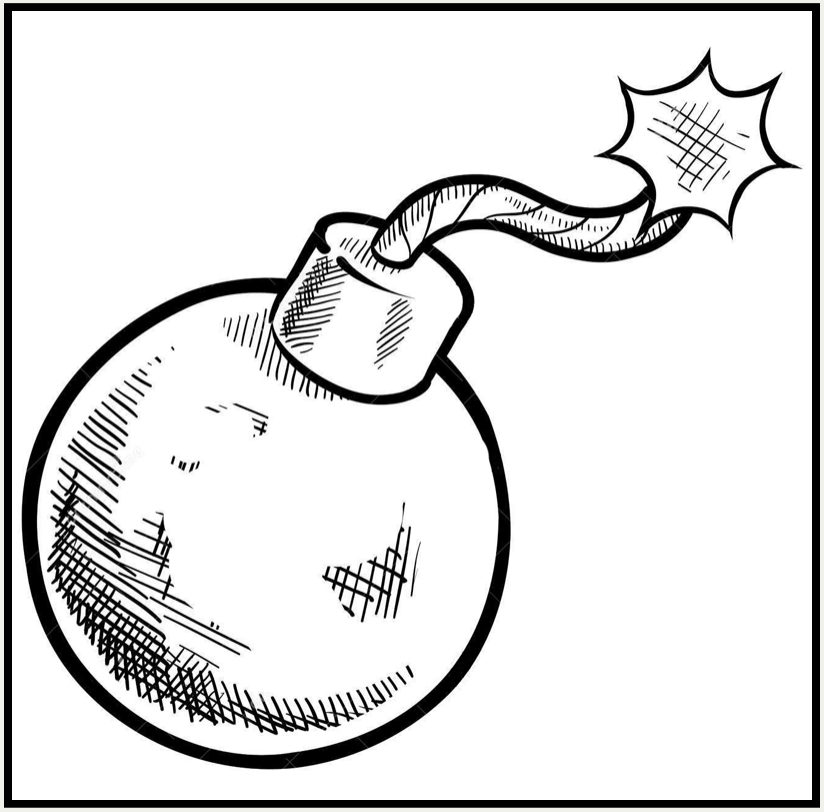
```
def add(a, b)
  a + b
end

puts add(10, 20)
puts add("foo", "bar")
```

```
30
foobar
```

```
def add(a, b)
  a + b
end

puts add(true, false)
```

```
Error in type_inference9.cr:5: instantiating 'add(Bool, Bool)'

puts add(true, false)
     ^~~

in type_inference9.cr:2: undefined method '+' for Bool

  a + b
    ^
```

Null Pointer Exception



The problem of nil value.

```
loop do
  word = gets
  break if word.upcase =~ /^QUIT/
  puts word.chomp.reverse
end
```

Let's run it with Ruby.

```
$ ruby nil.cr
foo
oof
bar
rab
^D
nil.cr:3:in `block in <main>': undefined method `upcase' for nil:NilClass (NoMethodError)
```

Crystal detects that kind of error during compilation phase.

```
$ crystal build nil.cr
in nil.cr:1: instantiating 'loop()'

loop do
^~~~

in nil.cr:3: undefined method 'upcase' for Nil (compile-time type is (String | Nil))

  break if word.upcase =~ /^QUIT/
               ^~~~~~
```

Let's fix it.

```
loop do
  word = gets
  break if word.nil? || word.upcase =~ /^QUIT/
  puts word.chomp.reverse
end
```

```
$ crystal run nil-fix.cr
foo
oof
bar
rab
^D
```

Performance

Crystal compiles into native binary using LLVM.

Expect it to be faster than Ruby.

How much?

What is the smallest number that is evenly divisible by all of the numbers from 1 to 20?

https://projecteuler.net/problem=5

Warning: naive solution

Ruby

```
puts 1.upto((2..20).reduce(:*))
  .each { |n| break n if (2..20).all? { |e| n % e == 0 } }
```

```
$ time ruby euler.rb
232792560
ruby euler.rb  115,36s user 0,80s system 95% cpu 2:01,88 total
```

Crystal

```
puts 1.upto((2..20).reduce(1_u64) { |acc, e| acc * e })
  .each { |n| break n if (2..20).all? { |e| n % e == 0 } }
```

```
$ crystal build --release euler.cr && time ./euler
232792560
./euler  1,65s user 0,01s system 98% cpu 1,679 total
```

Crystal faster ~72x

Kemal vs Sinatra

```
require "kemal"

logging false

get "/" do
  "Hello World"
end

Kemal.run
```

```
require "sinatra"

set :logging, false

get "/" do
  "Hello world"
end
```

wrk -c 100 -d 10 -t 1 http://localhost:3000

```
Requests/sec:  63258.85
Transfer/sec:  7.24MB
```

```
Requests/sec:  3344.73
Transfer/sec:  601.04KB
```

Crystal faster ~19x

Array#sort!

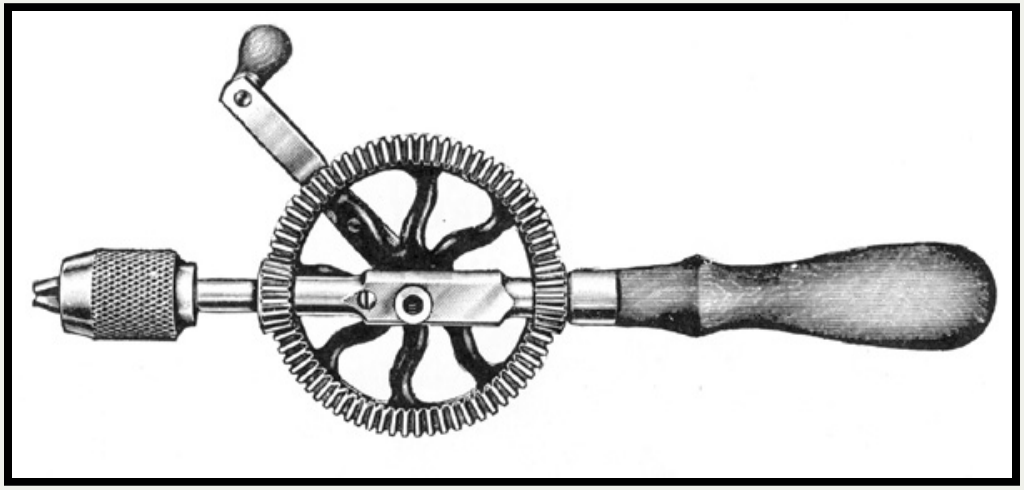
```
require "benchmark"

arr = (1..100_000_000).to_a.shuffle
puts Benchmark.measure { arr.sort! }
```

```
$ ruby array_bm.cr
 28.890000   1.190000  30.080000 ( 31.266588)
```

```
$ crystal run --release array_bm.cr
  9.890000   0.040000   9.930000 (  10.187163)
```

Crystal faster ~3x

Macros



```
{% puts "This runs during compilation" %}
puts "This runs during execution"
```

```
$ crystal build macro1.cr
"This runs during compilation"
```

```
$ ./macro1
This runs during execution
```

```
macro define_method(name, body)
  def {{name}}
    {{body}}
  end
end

class Foo
  define_method bar, "baz"

  # def bar
  #   "baz"
  # end
end
```

```
macro assert(assertion)
  unless {{assertion}}
    puts "Assertion {{assertion}} failed!"
  end
end

assert 1 == 2
```

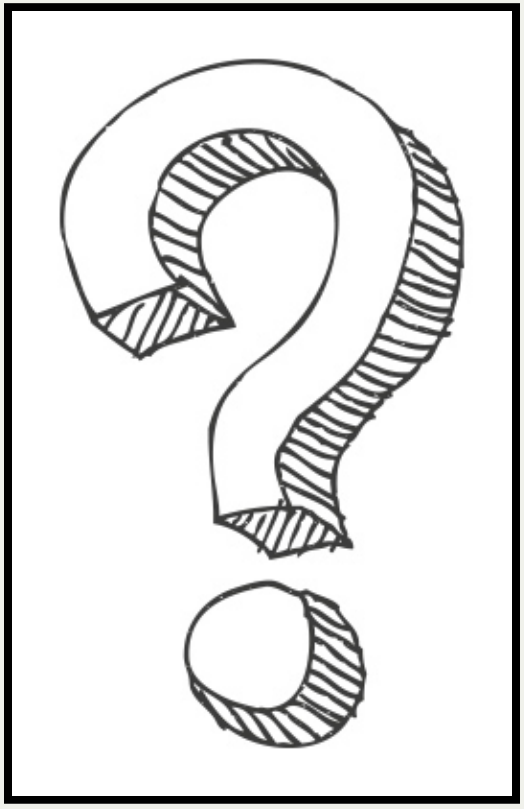
```
$ crystal run macro3.cr
Assertion 1 == 2 failed!
```

Macros enables metaprogramming.

```
class Person < ActiveRecord::Model
  adapter postgres

  table_name people

  primary id                    : Int
  field last_name               : String
  field first_name              : String
  field number_of_dependents    : Int
end
```

It's different from Ruby approach.

Summary



Is it worth to start programming in Crystal?

Looks like Ruby

Quacks like Ruby

Faster than Ruby

Catches more bugs earlier

It is Crystal

Happy crystaling!

Thank you!

Questions?