RUBY EASE; C SPEED

CRYSTAL LANGUAGE

RUBY EASE; C SPEED

- Crystal is a programming language that has a Ruby like syntax which is easy to read, clean and concise.
- Crystal is Statically typed and Compiled so it's very Efficient and provides C like performance (uses LLVM).
- Crystal uses Type Inference, Union Types, Macros, Lambda's and Closures to give you the feel of a interpreted language but performs like a natively compiled language.

RUBY EASE

```ruby
5.times do |i|
  puts i
end


[10, 20, 30].map { |x| x.to_s }


x = -1
puts x.abs


x = "hello"
puts x.size
```

RUBY EASE

- Classes, inheritance, modules, include, extend.
- Operator overloading.
- String, range, regex literals, heredoc, interpolation
- Multi-assignment (x, y = y, x)
- Blocks, procs and closures
- Big standard library: HTTP, OAuth, JSON, YAML, XML

RUBY EASE

```
class Person
  getter name
  property age

  def initialize(@name : String, @age : Int32)
  end
end

person = Person.new("Michael", 35)

person.name #=> "Michael" : String

person.age #=> 35 : Int32

person.age += 1 #=> 36 : Int32

person.name = "Other person" # undefined method 'name=' for Person
```

IN SOME CASES BETTER THAN RUBY

```
# arity and type based method overloading

class Dog
  def greet
    "Woof! Woof!"
  end

  def greet(name : String)
    "Woof #{name}!"
  end

  def greet(times : Int32)
    greet * times
  end
end
```

C SPEED

- Compare Crystal to Top Programming languages on Github
- JS, Java, Python, Ruby, PHP, C++, CSS, C#, C, Go
- Recursive Fibonacci Sequence (45)
- Slowest to Fastest
- https://github.com/drujensen/fib

#10 PHP – 6 MINUTES 2 SECONDS

```
<?php
function fib($n)
{
  if ($n <= 1) {
    return 1;
  } else {
    return fib($n - 1) + fib($n - 2);
  }
}

echo fib(45);
?>

$time php fib.php
1836311903
real 6m2.687s
```

#9 PYTHON - 5 MINUTES 44 SECONDS

```python
def fib(n):
    if n <= 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

print fib(45)
```

```
$time python fib.py
1836311903
real 5m44.837s
```

#8 RUBY – 2 MINUTES 4 SECONDS

```
def fib(n)
  if n <= 1
    1
  else
    fib(n - 1) + fib(n - 2)
  end
end

puts fib(45)

$time ruby fib.rb
1836311903
real 2m4.082s
```

#7 NODE - 12.76 SECONDS

```
var fib = function(n) {
  if (n <= 1) {
    return 1;
  } else {
    return fib(n - 1) + fib(n - 2);
  }
};

console.log(fib(45));

$time node fib.js
1836311903
real 0m12.760s
```

#6 C# – 7.166 SECONDS

```
using System;

public class Fib
{
  public static uint fib(uint n)
  {
    if (n <= 1)
    {
        return 1;
    }
    else
    {
        return fib(n - 1) + fib(n - 2);
    }
  }

  static void Main(string[] args)
  {
    Console.WriteLine(fib(45));
  }
}

$mcs fib.cs
$time mono fib.exe
1836311903
real 0m7.166s
```

#5 C - 6.999 SECONDS

```c
#include <stdio.h>

unsigned int fib(unsigned int n)
{
  if (n <= 1) {
    return 1;
  } else {
    return fib(n - 1) + fib(n - 2);
  }
}

int main(void) {
  printf("%d", fib(45));
  return 0;
}
```

```
$gcc -o fib fib.c
$time ./fib
1836311903
real 0m6.999s
```

#4 C++ - 6.969 SECONDS

```cpp
#include <iostream>

using namespace std;

int fib(int x) {
    if (x <= 1) {
        return 1;
    } else {
        return fib(x - 1) + fib(x - 2);
    }
}

int main()
{
    cout << fib(45);
}
```

```
$g++ -o fib fib.cpp
$time ./fib
1836311903
real 0m6.969s
```

#3 GO – 6.703 SECONDS

```
package main

import "fmt"

func fib(n uint) uint {
   if n <= 1 {
      return 1
   } else {
      return fib(n - 1) + fib(n - 2)
   }
}

func main() {
   fmt.Println(fib(uint(45)))
}

$go build fib.go
$time ./fib
1836311903
real 0m6.703s
```

#2 JAVA – 4.672 SECONDS

```java
import java.util.*;

public class Fib {
  static int fib(int n) {
    if (n <= 1)
      return 1;
    else
      return fib(n - 1) + fib(n - 2);
  }

  public static void main(String[] args) {
    System.out.print(fib(45));
  }
}
```

```
$javac Fib.java
$time java Fib
1836311903
real 0m4.672s
```

#1 CRYSTAL – 3.857 SECONDS

```
def fib(n)
  if n <= 1
    1
  else
    fib(n - 1) + fib(n - 2)
  end
end

puts fib(45)

$crystal build fib.cr -release
$time ./fib
1836311903
real 0m3.857s
```

BENCHMARK CAVEATS

- Crystal doesn't check for overflow with default Int32 datatype (yet) and will return a negative number.
- Test fib(46) which is larger than (2 ** 31) - 1
- C, C++, Crystal and Java do not check for overflow.
- Go, C#, Node, Ruby, Python and PHP do.

UNION TYPES

- Crystal supports union types.
- Unions allow a variable to be multiple types at the same time.
- Crystal determines the possible types at compile time using type inference.
- You need to cast to the specific type if not duck typed.

UNION TYPES EXAMPLE

```
def double(x : (Int32|String))
  x * 2
end

i = double(3)

puts i

s = double("hi")

puts s
```

TYPE INFERENCE

- Crystal is compiled and statically typed but leverages type inference to automatically determine the type during compilation.
- This allows the Crystal language to look similar to Ruby and avoid specifying the type for each variable.
- Once a type is determined, it sticks. Multiple types can be assigned to a single variable (unions).
- Empty Arrays and Hashes require types to be defined.

EXAMPLE TYPE INFERENCE

```
x = rand < 0.5 ? 1 : nil

case x
when Number
  puts x + 1
else
  puts "no x"
end
```

MACROS

- There are no dynamic meta-programming methods like method_missing found in Ruby.
- Crystal supports macros which can handle some situations similar to meta-programming.
- Macros are processed at compile-time but give you access to the AST nodes during compilation.
- Excellent for defining your own DSL.

EXAMPLE MACROS

```
macro rock(name)
  def {{name.id}}_rocks
    puts "{{name.id}} Rocks!"
  end
end


rock :crystal

crystal_rocks
```

C LIBRARIES

- Crystal makes it simple to map to C libraries
- Mapping is as simple as specifying a `lib` and `fun`
- Linking is automatic in most cases but you can specify the library `-l` if needed
- Other `-ldflags` are easily set as well

C FUN EXAMPLE

```
lib LibMath

  fun nearbyint(x: Float64): Float64

  fun pow(x: Float64, y: Float64): Float64

end



LibMath.nearbyint(3.534) #=> 4.0: Float64

LibMath.pow(2, 10) #=> 1024.0: Float64

#man 3 math
```

CONCURRENCY

- Crystal supports fibers using `spawn` and `channels` similar to Goroutines in GoLang.
- Fibers are light weight processes that have a very small memory footprint.
- Millions of fibers can be launched simultaneously.
- The main thread can `receive` results similar to `select` in GoLang.

CONCURRENCY EXAMPLE

```
require "http/client"
channel = Channel(String).new
spawn do
  channel.send(HTTP::Client.get("https://crystal-lang.org").body)
end
spawn do
  channel.send(HTTP::Client.get("https://ruby.org").body)
end
2.times do
  puts channel.receive
end
```

HIGHLIGHTS

- Everything is an Object
- Proc's, Blocks and Closures are supported
- Functional goodness
- Object Oriented goodness
- Easily integrates with C libraries using lib and fun
- Macros make it easy to create a DSL
- Concurrency capabilities similar to GoLang

REFERENCES

- crystal-lang.org
- http://www.techworm.net/2016/09/top-10-popular-programming-languages-github.html
- https://github.com/drujensen/fib
- https://citizen428.net/a-rubyist-looks-at-crystal-part-1-86a9284c936e#.u6p9t92av
- https://github.com/will/crystal_workbook
- https://github.com/crystal-lang/crystal-presents/releases/tag/2016.09-rubyconfbr