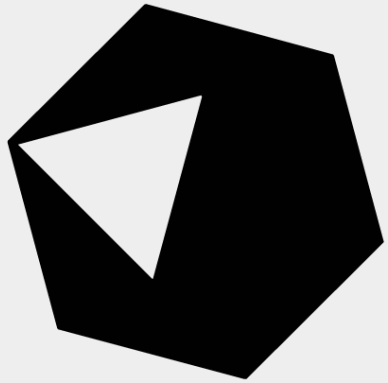


CRYSTAL

Programas eficientes sin resignar felicidad

@CrystalLanguage

Juan Edi

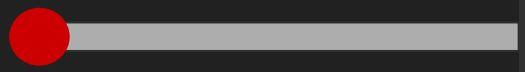
https://manas.tech

@juanedi

¿Por qué un lenguaje nuevo?

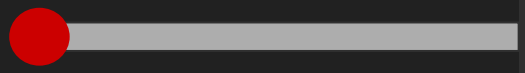
Breve (e incompleta) historia de los lenguajes de programación

Assembly



```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80
    mov     eax,1
    int     0x80

section     .data

msg     db  'edi x0x0 <xD',0xa
len     equ $ - msg
```

Assembly

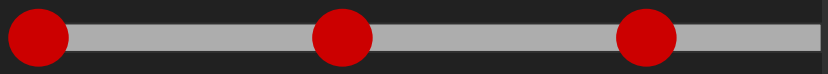


```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80
    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello world!',0xa
len     equ $ - msg
```

C

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  printf("Hello, world!");
  return EXIT_SUCCESS;
}
```

Java



```
public class HelloWorld {
  public static void main(String[] args) {
    Map<String, Integer> edades =
          new HashMap<String, Integer>();

    m.put("María", 23);
    m.put("Carlos", 18);

    for (Integer edad : m.values()) {
      System.out.println(edad);
    }
  }
}
```

Ruby

```
puts "Hello, world!"
```

Ruby

```ruby
edades = {
  "María" => 23,
  "Carlos" => 18
}

edades.values
      .each { |e| puts e }
```

¿Se puede tener todo?

- ➡ Los lenguajes compilados son más eficientes
- ➡ El chequeo de tipos nos da seguridad
- ➡ Los lenguajes dinámicos son muy prácticos

Crystal

Objetivos

Buscamos un balance distinto entre performance y *felicidad*

- ➡ Compilado
- ➡ Tipado estático
- ➡ Fácil de leer, escribir y aprender

Sintaxis inspirada en Ruby

```
# The Greeter class
class Greeter

  def initialize(@name : String)
  end

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

# Create a new object
g = Greeter.new("world")

# Output "Hello World!"
g.salute
```

- ➡ Poca ceremonia
- ➡ APIs sencillas
- ➡ Fácil de aprender

Chequeo estático de tipos

- ➡ Análisis tiempo de compilación
- ➡ Detección automática
- ➡ Testear lo necesario
- ➡ Refactors más seguros

```
x = 1
puts x

x.oops
```

```
undefined method 'oops' for Int32
```

Bajo nivel (generación de código nativo)

- Más fácil de distribuir
- Inicio más rápido
- Mejor uso de recursos
- Garbage collector

```
$ crystal build hello.cr

$ otool -Vt hello

…

leaq0x1e216(%rip), %rdi
callq   "_*puts<String>:Nil"
…
```

Bajo nivel (bindings a C)

```
@[Link(ldflags: "-lpq -L`pg_config --libdir`")]
lib LibPQ
  fun connect = PQconnectdb(conninfo : UInt8*) : Void*
  fun exec = PQexec(conn : Void*, query : UInt8*) : Void*
  fun getvalue = PQgetvalue(res : Void*, row : Int32, column :
Int32) : UInt8*
end

conn = LibPQ.connect("postgres:///")
q = "select 'Hello it is ' || now()"
res = LibPQ.exec(conn, q)
puts String.new(LibPQ.getvalue(res, 0, 0))
```

Curso rápido

- **Compilar y ejecutar programas**
- Sintaxis y semántica
- Sistema de tipos
- Metaprogramación

Curso rápido

- ➡ Compilar y ejecutar programas
- ➡ **Sintaxis y semántica**
- ➡ Sistema de tipos
- ➡ Metaprogramación

Literales

```
➡ true, false              # Bool
➡ nil                      # Nil
➡ 1, -4                    # Int32
➡ 1_i8, 1u16               # Int8, UInt16
➡ 1.0                      # Float64
➡ 1.0_f32                  # Float32
➡ ‘a’                      # Char
➡ “Hello world”            # String
➡ “result: #{a + b}”       # String
➡ :hello                   # Symbol
```

Literales

```
➡ [1,2,3]                          # Array(Int32)

➡ [1,”hello”,’x’]


➡ { 1 => ‘a’, 2 => ‘b’ }           # Hash(Int32, Char)

➡ { 1 => ‘a’, 2 => “hola”}
```

Estructuras de control

```
a = 1

if a > 0
  a = 10
end
```

```
b = 1

if b > 2
  b = 10
else
  b = 20
end
```

Estructuras de control

```
def validate(x : Int32)
  return :error if x < 4

  :ok
end
```

Estructuras de control

```
def validate_all(items : Array(Int32))
  i = 0
  ret = true

  while i < items.size
    ret = ret || validate(items[i])
    i += 1
  end

  ret
end
```

Objetos

```
# The Greeter class
class Greeter

  def initialize(@name : String)
  end

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

# Create a new object
g = Greeter.new("world")

# Output "Hello World!"
g.salute
```

Definición de la clase

Constructor

Método de instancia

Instanciación

Envío de mensajes

¿Qué tipo tienen estos elementos?

```
[1, 2, 3]

["crystal", "es", "genial"]
```

Generics

```
class Box(T)

  def initialize(@value : T)
  end

  def value
    @value
  end
end
```

Parámetro de tipo

```
int_box = Box.new(1)
int_box.value
```

Int32

```
string_box = Box.new("hello")
string_box.value
```

String

Bloques

```ruby
def twice
  yield
  yield
end
```

```ruby
twice do
  puts “Hello!”
end


# equivalente
twice { puts “Hello!” }
```

Bloques

```ruby
def twice
  yield 1
  yield 2
end
```

```ruby
twice do |i|
  puts i
end


# equivalente
twice { |i| puts i }
```

Bloques

```
def validate_all(items : Array(Int32))
  items.all? { |i| i >= 4 }
end
```

Bloques

```
items = [3,6,10]


items.all?   { |i| i >= 4 }      # false

items.select { |i| i >= 4 }      # [6,10]

items.map    { |i| i >= 4 }      # [false, true, true]
```

Bloques

```
read_file “rosario.txt” do |content|
  puts content.reverse
end
```

Curso rápido

- Compilar y ejecutar programas
- Sintaxis y semántica
- **Sistema de tipos**
- Metaprogramación

Inferencia

¿Qué argumentos puede recibir el siguiente método?

```
def inc(x)
  x + 1
end
```

```
inc(4)
```

Retorna 5

```
inc("cuatro")
```

Error de compilación

Inferencia

Típicamente los lenguajes estáticamente tipados se ven así:

```
def shade_pixel(ray : Ray, obj : Sphere, tval : Float64) : Int32
  pi : Vector = ray.orig + ray.dir.scale(tval)
  color : Color = diffuse_shading(pi, obj, LIGHT1)
  col : Float64 = (color.r + color.g + color.b) / 3.0
  (col * 6.0).to_i
end
```

Inferencia

```
def shade_pixel(ray      , obj          , tval          )
  pi          = ray.orig + ray.dir.scale(tval)
  color         = diffuse_shading(pi, obj, LIGHT1)
  col           = (color.r + color.g + color.b) / 3.0
  (col * 6.0).to_i
end
```

Inferencia

```
def shade_pixel(ray, obj, tval)
  pi = ray.orig + ray.dir.scale(tval)
  color = diffuse_shading(pi, obj, LIGHT1)
  col = (color.r + color.g + color.b) / 3.0
  (col * 6.0).to_i
end
```

Union types

```
if ARGV.empty?
  x = “hello”
else
  x = nil
end

puts x
```

¿Es válido este programa?

Union types

```
if ARGV.empty?
  x = “hello”
else
  x = nil
end

puts x.size
```

¿Y este?

```
Undefined method ‘size’ for Nil (compile-time type is
(String | Nil))
```

Curso rápido

- ➡ Compilar y ejecutar programas
- ➡ Sintaxis y semántica
- ➡ Sistema de tipos
- ➡ **Metaprogramación**

Metaprogramación

- *Código que modifica el programa*
- Bastante utilizado en lenguajes dinámicos
- La experiencia suele no ser tan feliz en lenguajes tipados
- En Crystal se logra mediante **macros**

Macros

```
class Person
  def initialize(@name : String, @age : String)
  end

  def name
    @name
  end

  def age
    @age
  end
end
```

¿Qué hay que hacer para agregar una propiedad nueva?

Macros

```
class Person
  public_props({
    “name” => String,
    “age” => Int32,
  })
end
```

¿Qué hay que hacer para agregar una propiedad nueva?

Macros

```
macro public_props(properties)
  def initialize({% for name, type in properties %}
                   @{{name.id}} : {{type}},
                 {% end %})
  end

  {% for name, type in properties %}
    def {{name.id}}
      @{{name.id}}
    end
  {% end %}
end
```

Estado del proyecto

Roadmap

- ➡ Paralelismo
- ➡ Pulir librería estándar y documentación
- ➡ Soporte para otras plataformas (ARM, Windows)
- ➡ Debugger

Comunidad

- Lista de correo, IRC/Gitter, Twitter
- +6000 estrellas en GitHub
- Meetups en Europa, Asia y América
- Documentación en inglés, ruso, portugués, japonés y taiwanés
- +100 colaboradores en el compilador, librería estándar, documentación y website
- +70 sponsors, desde 1 USD por mes

¿Cómo contribuir?

- ➡ Documentación, pull requests, issues
- ➡ Bountysource
- ➡ Ayudar con herramientas y soporte de editores
- ➡ Escribir código Crystal
- ➡ Compartir

¿Preguntas?

¡A trabajar!

- ➡ git clone git://labdcc.fceia.unr.edu.ar/pub/jcc/crystal.git
- ➡ cd crystal/exercism
- ➡ crystal spec 01-hello-world