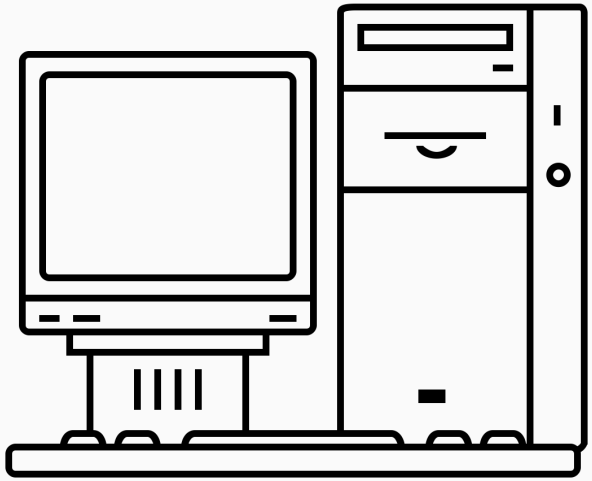
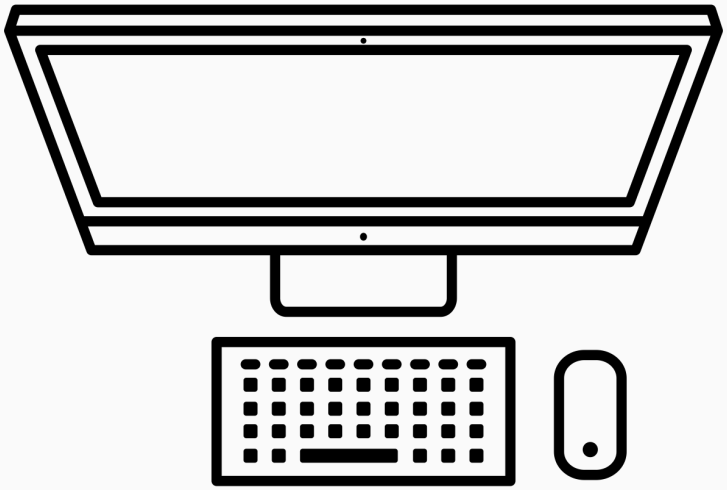
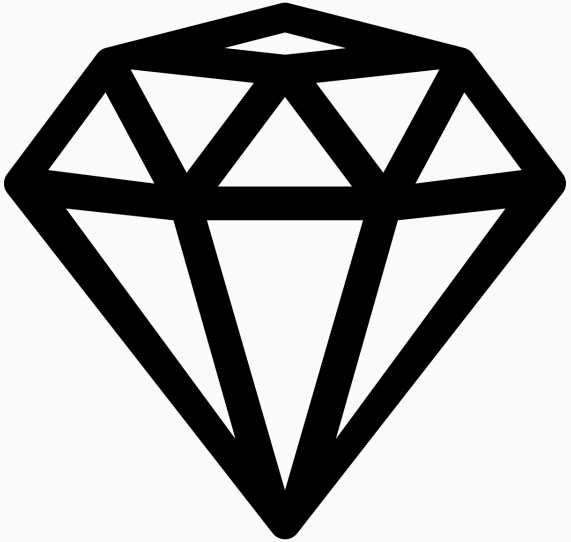
Crystal

O mundo de 15 anos atrás

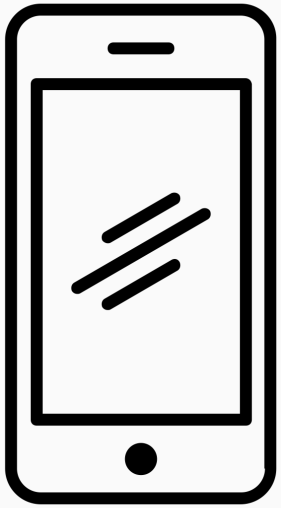


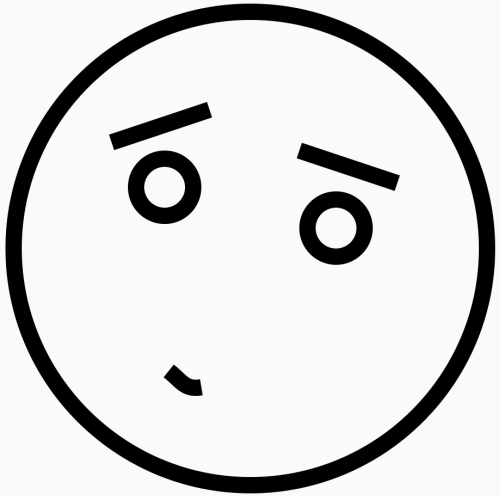
O mundo de 10 anos atrás



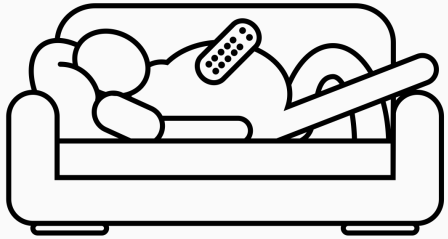


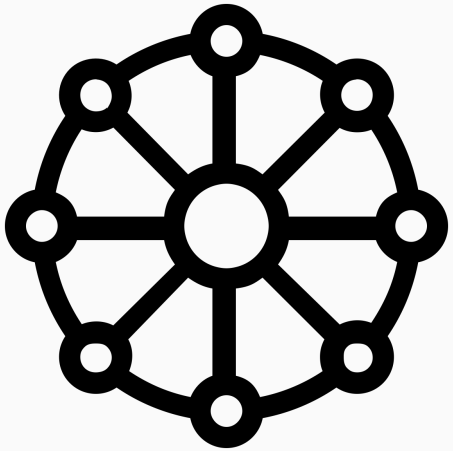
O mundo de hoje



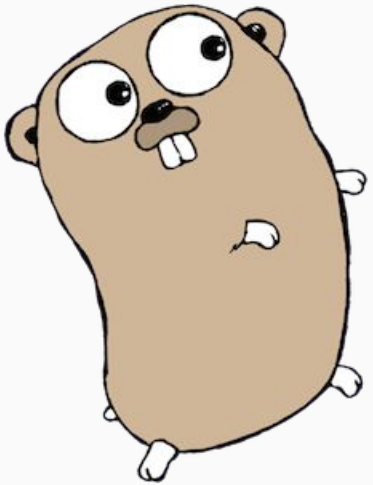


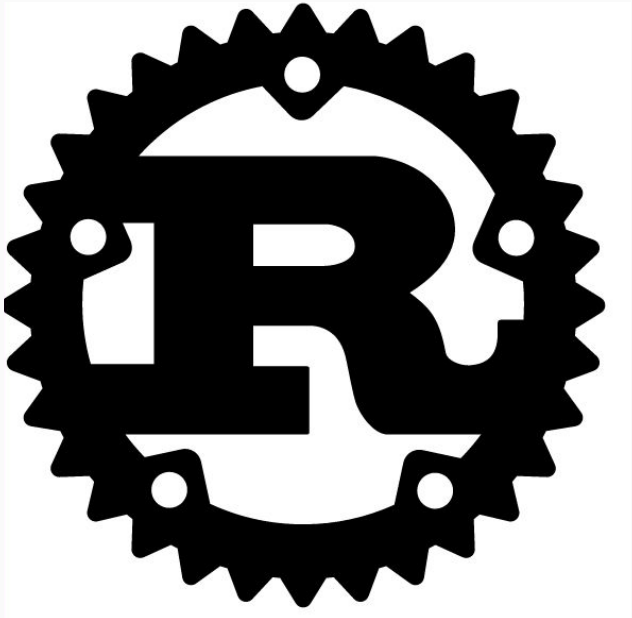
Outras preocupações





Os últimos cinco anos





LLVM é rei

Crystal

Este é um programa em Crystal

```
def average_age(people)
  return 0 if people.empty?
  sum_ages = people.map { |p| p.age }.reduce(0) { |a, b| a + b }
  sum_ages / people.size
end


puts average_age([Person.new("Mary", 10), Person.new("John", 13)])
```

E é assim que você o roda

```
$ crystal average_age.cr
11

$ crystal build average_age.cr

$ ./average_age

11
```

Tipos estáticos

```
def average_age(people)
  return 0 if people.empty?
  sum_ages = people.map { |p| p.age }.reduce(0) { |a, b| a + b }
  sum_ages / people.size
end


puts average_age([Person.new("Mary", “10”), Person.new("John", “10”)])
```

Vida real

```
$ crystal build average_age.cr

...

in ./average_age.cr:11: no overload matches 'Int32#+' with type String
Overloads are:
 - Int32#+(other : ::Int8)
 - Int32#+(other : ::Int16)
 - Int32#+(other : ::Int32)

...

sum_ages = people.map { |p| p.age }.reduce(0) { |a, b| a + b }
```

Anotações de tipos

```
def average_age(people : Array(Person))
  return 0 if people.empty?
  sum_ages = people.map { |p| p.age }.reduce(0) { |a, b| a + b }
  sum_ages / people.size
end


puts average_age([Person.new("Mary", 10), Person.new("John", 13)])
```

Anotações de tipos

```
def average_age(people : Array(Person)) : Int32
  return 0 if people.empty?
  sum_ages = people.map { |p| p.age }.reduce(0) { |a, b| a + b }
  sum_ages / people.size
end

puts average_age([Person.new("Mary", 10), Person.new("John", 13)])
```

Inferência de tipos

```
def magic_size(obj)
 obj.size * 1337
end


puts magic_size(“foo”)
puts magic_size([“foo”, “bar”, “baz”])
```

Inferência de tipos

```
def magic_size(obj)
 obj.size * 1337
end


puts magic_size(“foo”)
puts magic_size([“foo”, “bar”, “baz”])
```

Inferência de tipos

```
def magic_size(obj)
 obj.size * 1337
end


puts magic_size(“foo”)
puts magic_size([“foo”, “bar”, “baz”])
```

Union Types

```
def magic_size(obj) # magic_size(obj : (String | Array))
 obj.size * 1337
end


puts magic_size(“foo”)
puts magic_size([“foo”, “bar”, “baz”])
```

Bindings molezinha para C

```
@[Link("fast_triangulation")]
lib LibFastTriangulation
  type Faces = Int32*
  type Vertices = Float64*
  # int triangulate(float64_t *const vertices, int32_t **faces)
  fun triangulate(vertices: Vertices, faces: Faces*) : Int32
end
```

Performance supimpa

DEMO

Concorrência e Paralelismo

- Primitiva `Fiber + CSP` via `Channels`
- Scheduler decide melhor momento de execução
- Por enquanto scheduler usa apenas um thread nativo, mas estão trabalhando nisto
- Tudo que é IO é automaticamente executado em uma `Fiber`

Para mais informações

http://crystal-lang.org/

https://github.com/veelenga/awesome-crystal

Freenode: #crystal-lang