Do Zero ao Julia

João Víctor Costa de Oliveira

2022

Conteúdos abordados

Introdução

- Linguagem de programação
  - Dinâmica
  - Rápida
  - Estruturada
  - Compilada JIT
  - OpenSource

Introdução

- Nascida em fevereiro de 2012
- Pensada para o uso em computação científica
- Criada para unir o melhor das linguagens
  - Dinamismo do Ruby
  - Notação matemática do Matlab
  - Aplicado a estatísticas como o R
  - Simples como Python
  - Rápido como C

Introdução

Filosofia Julia

FILOSOFIA JULIA

Ler como Python e rodar como C

Introdução

Empresas, instituições e projetos que usam Julia

- INPE
- LAMPS - PUC Rio
- IBM [1]
- MIT
- CISCO
- BNDES

[1] em um dos projetos no qual utilizam Julia, uma Rede Neural teve um aumento de velocidade de processamento em torno de 57%

Tipos de dados

Tipos de dados

Inteiro (10, -2, 4523, ...)
Ponto flutuante (simples) (10.0, 4.6, 7.8954, ...)
Complexo (4.5 + 3.78i = complex(4.5, 3.2)
Booleanos (True, False)
String (“abc”, “Fluminense”, ...)

Tipos de dados

Tipagem de variáveis

- Tipagem
  - dinâmica
  - forte
- Variáveis não precisam ter o tipo declarado
- Quem cuida de atribuir um tipo para cada variável é o próprio compilador

```
a = 12
b = 15.78
c = """Joffrey 'Velaryon' """
d = complex(4.785, 3.008)

println(typeof(a),"    ", typeof(b),"    ", typeof(c),"   ", typeof(d))
```

```
Int64    Float64    String   ComplexF64
```

Tipos de dados

Tipagem de variáveis

- Podemos ainda criar uma variável como inteiro, e depois atribuí-la um valor float, por exemplo

```
a = 12
println(typeof(a))
a = 12.56879
println(typeof(a))
```

```
Int64
Float64
```

Tipos de dados

Declaração de variáveis

- Declaramos variáveis em Julia assim como declaramos em várias outras linguagens, usando o sinal de " = "
- Ao contrário de linguagens como *C* e *C++*, o ";"ao final de cada linha não é necessário

```
nome = "Eddard"
casa = "Stark"
nobre = True
quantidade_de_cabecas = 0
primeiro_filho, segundo_filho = "Robb", "Bran"
```

Tipos de dados

Declaração de variáveis

- Julia nos permite escrever códigos super genéricos
- Podemos escrever um sistema todo nomeando as variáveis com emojis ou símbolos
- Isso pode ser interessante quando queremos escrever expressões por exemplo

```
🪓 = -1
😄 = 1
😵 = 0;
```

```
println("Eddard Stark se recusou a ajoelhar ao Rei Joffrey I Baratheon ordenou que:")
😄 + 🪓 == 😵
```

```
Eddard Stark se recusou a ajoelhar ao Rei Joffrey I Baratheon ordenou que:

true
```

Tipos de dados

Função de impressão

- Utilizamos a função “ println() "para imprimir com quebra de linha ao final
- Utilizamos a função “ print() "para imprimir sem quebra de linha ao final

```
nome = "Eddard"
casa = "Stark"
qtd_filhos_homens = 3

println("Lord $nome casa $casa teve $qtd_filhos_homens filhos homens legítimos")
print("A casa $casa é uma das mais tradicionais dos Sete Reinos, e descendente dos Primeiros Homens")
println(" assim como a maioria das casas do Norte.")
```

```
Lord Eddard casa Stark teve 3 filhos homens legítimos
A casa Stark é uma das mais tradicionais dos Sete Reinos, e descendente dos Primeiros Homens assim como a maioria
das casas do Norte.
```

Tipos de dados

Pegando valores do teclado

- Usamos a função “ readline() "para ler um input do teclado
- Como para Julia tudo que vem do teclado é uma string, precisamos fazer um typecasting
- Para isso, usamos a função reservada “ parse() "

```
numero = readline()

println(typeof(numero))

numero = parse(Int64, numero)

println(typeof(numero))
```

```
stdin> 55
String
Int64
```

Tipos de dados

Operações com variáveis numéricas

Operação	Sintaxe	Operação	Sintaxe
Adição	a + b	Valor absoluto	abs(a)
Subtração	a - b	Converter em inteiro	convert(Int64, variavel)
Produto	a * b	Converter em ponto flutuante	convert(Float64, variavel)
Divisão	a / b	Potenciação	a ^ b
Divisão Inteira	a // b		
Módulo	a % b		
Negação	- a		

Tipos de dados

Trabalhando com strings

- Em Julia podemos inicializar strings de duas maneiras

```
s1 = "Eu sou uma string"
```

```
"Eu sou uma string"
```

```
s2 = """Eu também sou uma string"""
```

```
"Eu também sou uma string"
```

Tipos de dados

Trabalhando com strings

- Geralmente usamos a segunda maneira quando precisamos colocar aspas dentro da string
- Isso pode ocorrer ao fazermos uma citação, por exemplo

```
"Aqui, teremos uma mensagem de "erro" ao tentar usar aspas dentro do texto, pois se torna ambiguo onde a string termina"
```

```
syntax: cannot juxtapose string literal

Stacktrace:
 [1] top-level scope
   @ In[3]:1
 [2] eval
   @ ./boot.jl:368 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1428
```

```
"""Já aqui, podemos usar "aspas" normalmente!!!"""
```

```
"Já aqui, podemos usar \"aspas\" normalmente!!!"
```

Tipos de dados

Trabalhando com strings

Atenção

Não confunda ‘a’ com "a". Um é um caracter e o outro é uma string

```
typeof('a')
```

```
Char
```

Tipos de dados

Interpolando strings

- Podemos usar o sinal $ para inserir variáveis existentes em uma string e avaliar expressões com a string

```
nome = "Joao Victor"
idade = 21
numero_de_matricula = "202065139B"
numero_de_dedos_da_mao = 10
numero_de_dedos_do_pe = 10
```

```
10
```

```
println("Olá, meu nome é $nome")
println("Eu tenho $idade anos de idade e meu numero de matricula eh $numero_de_matricula")
```

```
Olá, meu nome é Joao Victor
Eu tenho 21 anos de idade e meu numero de matricula eh 202065139B
```

```
println("Ao todo eu tenho $(numero_de_dedos_da_mao + numero_de_dedos_do_pe) dedos!")
```

```
Ao todo eu tenho 20 dedos!
```

Tipos de dados

Concatenação de strings

- Vamos ver três maneiras de concatenar strings
- A primeira é usar a função string()
- string() também converte entradas “não-string"em strings

```
s3 = "Quantos cachorros "
s4 = "são muitos cachorros?"
🐶 = 18
```

```
18
```

```
string(s3, s4)
```

```
"Quantos cachorros são muitos cachorros?"
```

```
string("Eu não sei, mas ", 🐶, " é muito pouco!")
```

```
"Eu não sei, mas 18 é muito pouco!"
```

Tipos de dados

Concatenação de strings

- Também podemos usar * para concatenação

```
s3*s4
```

```
"Quantos cachorros são muitos cachorros?"
```

Estruturas de Dados

Introdução

- Vamos começar agora a trabalhar com Estruturas de Dados que podem armazenar mais de um valor
- Vamos ver as mais básicas por enquanto
  - Tuplas
  - Dicionários
  - Arrays

Estruturas de Dados

Introdução

- Essas já são implementadas nativamente na linguagem
- Um pequeno spoiler
  - Tuplas e arrays são sequências ordenadas de elementos
  - Dicionários e arrays são mutáveis

Estruturas de Dados

Antes de começarmos...

Em Julia, usamos a notação indicial começada em 1 em vez de 0

Estruturas de Dados

Tuplas

- Tuplas são coleções de itens que não necessariamente são do mesmo tipo
- Não são mutáveis após a criação

Estruturas de Dados

Tuplas - Declaração

```
meus_animais_favoritos = ("cachorro", "leão", "urso")
```

```
("cachorro", "leão", "urso")
```

Estruturas de Dados

Tuplas - Acesso

```
meus_animais_favoritos[1]
```

```
"cachorro"
```

Estruturas de Dados

Tuplas

- Se tentamos mudar um elemento expecífico de uma tupla

```
meus_animais_favoritos[2] = "baleia"
```

```
MethodError: no method matching setindex!(::Tuple{String, String, String}, ::String, ::Int64)

Stacktrace:
 [1] top-level scope
   @ In[3]:1
 [2] eval
   @ ./boot.jl:368 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1428
```

Estruturas de Dados

Tuplas

- Podemos usar a palavra-chave “in"para checar se um elemento pertence a uma tupla

```
"cachorro" in meus_animais_favoritos
```

```
true
```

Estruturas de Dados

Tuplas-Nomeadas

- Tuplas-Nomeadas são semelhantes a Tuplas
- A exceção é que cada elemento tem um nome adicional

Estruturas de Dados

Tuplas-Nomeadas

```
meus_animais_favoritos = (mamifero = "cachorro", ave = "pinguim", marsupiais = "coala")
```

```
(mamifero = "cachorro", ave = "pinguim", marsupiais = "coala")
```

Estruturas de Dados

Tuplas-Nomeadas

- Também podemos acessar os elementos via indexação

```
meus_animais_favoritos[1]
```

```
"cachorro"
```

Estruturas de Dados

Tuplas-Nomeadas

- Ganhamos uma nova maneira de acessar os elementos

```
meus_animais_favoritos.marsupiais
```

```
"coala"
```

Estruturas de Dados

Dicionários

- Usamos dicionários quando temos um conjunto de dados que se relacionam
- Um bom exemplo para um dicionário é a lista de contatos de emails

Estruturas de Dados

Dicionários

```
minha_lista_de_emails = Dict("Stenio" => "stenio.soares@ufjf.br", "Camata" => "jose.camata@ufjf.br", "Ruy" => "ruy.reis@ufjf.br")
```

```
Dict{String, String} with 3 entries:
  "Camata" => "jose.camata@ufjf.br"
  "Ruy"    => "ruy.reis@ufjf.br"
  "Stenio" => "stenio.soares@ufjf.br"
```

Estruturas de Dados

Dicionários

- Podemos acessar nossos elementos por meio da sua chave

```
minha_lista_de_emails["Ruy"]
```

```
"ruy.reis@ufjf.br"
```

Estruturas de Dados

Dicionários

- Podemos adicionar novos elementos no dicionário

```
minha_lista_de_emails["Flávia"] = "flavia.bastos@ufjf.br"
```

```
"flavia.bastos@ufjf.br"
```

```
minha_lista_de_emails
```

```
Dict{String, String} with 4 entries:
  "Camata" => "jose.camata@ufjf.br"
  "Ruy"    => "ruy.reis@ufjf.br"
  "Stenio" => "stenio.soares@ufjf.br"
  "Flávia" => "flavia.bastos@ufjf.br"
```

Estruturas de Dados

Dicionários

- E também deletar elementos do dicionário

```
pop!(minha_lista_de_emails, "Camata")
```

```
"jose.camata@ufjf.br"
```

```
minha_lista_de_emails
```

```
Dict{String, String} with 3 entries:
  "Ruy"    => "ruy.reis@ufjf.br"
  "Stenio" => "stenio.soares@ufjf.br"
  "Flávia" => "flavia.bastos@ufjf.br"
```

Estruturas de Dados

Dicionários

- Ao contrário de Tuplas e Arrays, dicionários não são ordenados
- Assim, não conseguimos acessá-los por índices

```
minha_lista_de_emails[1]
```

```
KeyError: key 1 not found

Stacktrace:
 [1] getindex(h::Dict{String, String}, key::Int64)
   @ Base ./dict.jl:498
 [2] top-level scope
   @ In[13]:1
 [3] eval
   @ ./boot.jl:368 [inlined]
 [4] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1428
```

Estruturas de Dados

Dicionários

- Assim como em dicionários da vida real, em Julia seguimos a ordem alfabética das chaves

Estruturas de Dados

Arrays

- Diferente de Tuplas, arrays são estruturas de dados mutáveis
- Diferente de dicionários, arrays contém coleções ordenadas

Estruturas de Dados

Arrays - Declaração

```
disciplinas = ["Física III", "Sistemas Operacionais", "Solução Numérica de Equações Diferenciais", "Seminário em Computação V"]
```

```
4-element Vector{String}:
 "Física III"
 "Sistemas Operacionais"
 "Solução Numérica de Equações Diferenciais"
 "Seminário em Computação V"
```

Estruturas de Dados

Arrays - Declaração

```
fibonacci = [1, 1, 2, 3, 5, 8, 13]
```

```
7-element Vector{Int64}:
  1
  1
  2
  3
  5
  8
 13
```

Estruturas de Dados

Arrays - Declaração

```
mistura = [1, 1, 2, 3, "Sistemas Operacionais", "Física III"]
```

```
6-element Vector{Any}:
 1
 1
 2
 3
  "Sistemas Operacionais"
  "Física III"
```

Estruturas de Dados

Arrays - Manipulação

- Uma vez declarado, podemos acessar um elemento específico do array

```
fibonacci[4]
```

```
3
```

Estruturas de Dados

Arrays - Manipulação

- Podemos também manipular seus elementos

```
mistura[6] = 7

mistura
```

```
6-element Vector{Any}:
 1
 1
 2
 3
  "Sistemas Operacionais"
 7
```

Estruturas de Dados

Arrays - Manipulação

```
push!(fibonacci, 21)
```

```
8-element Vector{Int64}:
  1
  1
  2
  3
  5
  8
 13
 21
```

Estruturas de Dados

Arrays - Manipulação

```
pop!(fibonacci)
```

```
21
```

Estruturas de Dados

Arrays Multidimensionais

- Até agora trabalhamos com arrays unidimensionais
- Julia também nos permite trabalhar com arrays com um número arbitrário de dimensões

Estruturas de Dados

Arrays Multidimensionais

```
numeros = [[1,2,3],[4,5],[6,7,8,9]]
```

```
3-element Vector{Vector{Int64}}:
 [1, 2, 3]
 [4, 5]
 [6, 7, 8, 9]
```

Estruturas de Dados

Arrays Multidimensionais

- Também podemos inicializar aleatoriamente nossos arrays

Estruturas de Dados

Arrays Multidimensionais

```
#linhas, colunas
rand(4, 3)
```

```
4×3 Matrix{Float64}:
 0.51889   0.481311   0.854906
 0.581144  0.295459   0.261925
 0.89379   0.230699   0.72696
 0.618512  0.0630981  0.10119
```

Estruturas de Dados

Arrays Multidimensionais

```
rand(4, 3, 2)
```

```
4×3×2 Array{Float64, 3}:
[:, :, 1] =
 0.152255  0.0669432  0.470615
 0.273848  0.196422   0.916038
 0.317129  0.0583754  0.294517
 0.253621  0.359138   0.465867

[:, :, 2] =
 0.622617   0.463286  0.961333
 0.820463   0.290369  0.661976
 0.0190415  0.582667  0.55706
 0.523745   0.216207  0.93296
```

Estruturas de Dados

Arrays - Cópias

- Temos que ter muito cuidado ao fazer cópias de arrays

Estruturas de Dados

Arrays - Cópias

```
alguns_numeros = fibonacci
```

```
7-element Vector{Int64}:
  1
  1
  2
  3
  5
  8
 13
```

```
alguns_numeros[1] = 404
```

```
404
```

```
fibonacci
```

```
7-element Vector{Int64}:
 404
   1
   2
   3
   5
   8
  13
```

Estruturas de Dados

Arrays - Cópias

```
mais_alguns_numeros = copy(fibonacci)
```

```
7-element Vector{Int64}:
 404
   1
   2
   3
   5
   8
  13
```

```
fibonacci[1] = 1
fibonacci
```

```
7-element Vector{Int64}:
  1
  1
  2
  3
  5
  8
 13
```

```
mais_alguns_numeros
```

```
7-element Vector{Int64}:
 404
   1
   2
   3
   5
   8
  13
```

Blocos de identação

- Ao contrário de linguagens como C, C++ e Java, Julia não tem blocos de identação
- Sua “orientação"é bem parecida com a sintaxe do Matlab, onde não há uma obrigatoriedade de identar seu código

Blocos de identação

- Mas lembre-se: “Um código bem identado é um código mais legível!"

Condicionais

Sintaxe

- Como em várias linguagens, fazemos condicionais com a palavra-chave *if*

```
#=
if *condição1*
    *opção1*
elseif *condição2*
    *opção2*
else
    *opção3*
end
=#
```

Condicionais

Sintaxe

- Condicionais com a sintaxe acima nos permitem avaliar condicionalmente expressões
- Um bom caso para exemplo é o *Teste FizzBuzz*

Condicionais

Exemplo - Teste FizzBuzz

Dado um número, N, imprima "Fizz"se N é divisível por 3 , "Buzz"se N é divisível por 5 e "FizzBuzz"se é divisível por 3 e 5 ao mesmo tempo. Se nenhuma das condições for atendida, apenas imprima o número fornecido.

Condicionais

Exemplo - Teste FizzBuzz

```
N = 5
```

```
5
```

```
if (N % 3 == 0) && (N % 5 == 0)
    println("FizzBuzz")
elseif N % 3 == 0
    println("Fizz")
elseif N % 5 == 0
    println("Buzz")
else
    println(N)
end
```

```
Buzz
```

Condicionais

Operadores ternários

- Assim como em outras linguagens, Julia nos permite usar **operadores ternários**
- Funciona com o mesmo conceito que o *if...else*
- Obrigatoriamente precisamos devolver um valor após o teste
- Só é válido para operações que podem ser implementadas em uma única linha

Condicionais

Operadores ternários - Exemplo

- Se temos as seguintes variáveis

```
x = 10
y = 30
```

Condicionais

Operadores ternários - Exemplo

- Um código simples para saber qual delas é maior pode ser escrito com

```
if x < y
    println(x)
else
    println(y)
end
```

```
10
```

Condicionais

Operadores ternários - Exemplo

- Imagine que seu chefe te deu a seguinte ordem:

  "Você tem $R\$15000,00$ para esse projeto. A cada linha que vc escreve $R\$5,00$ são tirados do orçamento. Você tem que escrever todo o código de qualquer maneira, e se faltar orçamento, sai do seu salário!"

Condicionais

Operadores ternários - Exemplo

- Para não perdermos orçamento, e, provavelmente salário, podemos fazer o uso de Operadores Ternários

Condicionais

Operadores ternários - Exemplo

```
(x > y) ? x : y
```

```
30
```

Condicionais

Operadores ternários - Exemplo

- Economizamos 4 linhas de código
- Claro que esse foi um exemplo bobo e não real
- Mas operadores ternários são bem úteis no dia-a-dia

Estruturas de repetição

Sintaxe

```
#=
while *condição*
    *corpo do loop*
end
=#
```

Estruturas de repetição

Sintaxe

```
#=
for *var* in *contador*
    *corpo do loop*
end
=#
```

Estruturas de repetição

Exemplo

```
meus_professores = ["Ruy", "Luciano", "Stênio", "Golliat", "Rodrigo"]

i = 1

while i <= length(meus_professores)
    professor = meus_professores[i]
    println("Olá professor $professor.")
    i += 1
end
```

```
Olá professor Ruy.
Olá professor Luciano.
Olá professor Stênio.
Olá professor Golliat.
Olá professor Rodrigo.
```

Estruturas de repetição

Exemplo

```
# Primeiro, vamos inicializar o array com zeros

m, n = 5, 5
A = fill(0, (m, n))
```

```
# Irei usar uma iteração via column-major loop, para obter um melhor desempenho.
for j in 1:n
    for i in 1:m
        A[i, j] = i + j
    end
end

A
```

```
5×5 Matrix{Int64}:
 2  3  4  5   6
 3  4  5  6   7
 4  5  6  7   8
 5  6  7  8   9
 6  7  8  9  10
```

Exercício

Crie uma matriz antissimétrica de ordem 5. Após isso, imprima a matriz.

Exercício

Gabarito

```
A = fill(0, (5, 5));
```

```
for i in 1:5
    for j in 1:i
        (i == j) ? A[i, j] = 0 : A[i, j] = A[j, i] = (i+j)*((-1)^i)
    end
end

A
```

```
5×5 Matrix{Int64}:
  0   3  -4   5  -6
  3   0  -5   6  -7
 -4  -5   0   7  -8
  5   6   7   0  -9
 -6  -7  -8  -9   0
```

Funções

Como declarar uma função

- Em Julia, podemos declarar funções de algumas maneiras diferentes
- A primeira e mais comum irá requerer as palavras-chave *function* e *end*

Funções

Como declarar uma função

```
function digaoi(nome)
    println("Olá $nome, é um prazer te conhecer!")
end
```

```
digaoi (generic function with 1 method)
```

Funções

Como declarar uma função

```
function f(x)
    x^2
end
```

```
f (generic function with 1 method)
```

Funções

Como declarar uma função

- Podemos chamar nossas funções da seguinte maneira

```
digaoi("João Víctor")
```

```
Olá João Víctor, é um prazer te conhecer!
```

```
f(50)
```

```
2500
```

Funções

Duck-typing

"If it quacks like a duck, walks like a duck and looks like a duck, IT'S A DUCK!"

Funções

Duck-typing

- Nesse tipo de função o compilador irá trabalhar apenas com as entradas que fazem algum sentido
- Por exemplo, nossa função *digaoi()* funciona com qualquer tipo de dado

```
digaoi(1254789)
```

```
Olá 1254789, é um prazer te conhecer!
```

Funções

Duck-typing

- A função *f()* também funciona com matrizes

```
A = rand(3, 3)
A
```

```
3×3 Matrix{Float64}:
 0.48938   0.891262   0.902714
 0.597624  0.0366211  0.317802
 0.657349  0.589053   0.541296
```

```
f(A)
```

```
3×3 Matrix{Float64}:
 1.36553   1.00055   1.21365
 0.523257  0.721183  0.723147
 1.02955   0.926294  1.0736
```

Funções

Duck-typing

- Isso acontece pois A^2 é algo bem definido
- O que não acontece por exemplo com um vetor

```
v = rand(3)
v
```

```
3-element Vector{Float64}:
 0.8064779424371038
 0.30789741978681695
 0.4450167430060278
```

```
# NÃO IRÁ FUNCIONAR
f(v)
```

Structs

Sintaxe

```
#=
struct 'nome_da_struct'
    'código'
end
=#
```

Structs

Sintaxe

```
struct MeuTipo
    campo_1
    campo_2
end
```

Structs

Sintaxe

```
a = MeuTipo("Olá ", "mundo!")
```

```
MeuTipo("Olá ", "mundo!")
```

Structs

Sintaxe

```
mutable struct Pessoaa
    nome::String
    idade::Float64
end
```

```
Joao = Pessoaa("João", 21)
```

```
Pessoaa("João", 21.0)
```

Structs

Sintaxe

```
Joao.idade += 1
```

```
22.0
```

Bibliotecas

Pacotes

- Julia tem mais de 2000 pacotes registrados
- A maioria dos pacotes estão registrados no próprio site da linguagem [2]

[2] https://julialang.org/packages/

Bibliotecas

Instalando pacotes

A primeira vez que usamos um pacote em uma instalação nova da linguagem Julia, precisamos pedir ao gerenciador de pacores que adicione-o explicitamente

```
using Pkg
Pkg.add("Example")
```

```
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
   Installed Example ─ v0.5.3
    Updating `~/.julia/environments/v1.8/Project.toml`
  [7876af07] + Example v0.5.3
    Updating `~/.julia/environments/v1.8/Manifest.toml`
  [7876af07] + Example v0.5.3
Precompiling project...
  ✓ Example
  1 dependency successfully precompiled in 0 seconds. 27 already precompiled. 1 skipped during auto due to previous errors.
```

Bibliotecas

Exemplo

Um pacote legal para brincar é o “Colors"

```
Pkg.add("Colors")
```

```
   Resolving package versions...
   Installed FixedPointNumbers ─ v0.8.4
   Installed ColorTypes ──────── v0.11.4
   Installed Reexport ────────── v1.2.2
   Installed Colors ──────────── v0.12.8
    Updating `~/.julia/environments/v1.8/Project.toml`
  [5ae59095] + Colors v0.12.8
    Updating `~/.julia/environments/v1.8/Manifest.toml`
  [3da002f7] + ColorTypes v0.11.4
  [5ae59095] + Colors v0.12.8
  [53c48c17] + FixedPointNumbers v0.8.4
  [189a3867] + Reexport v1.2.2
  [2f01184e] + SparseArrays
  [10745b16] + Statistics
Precompiling project...
  ✓ Reexport
  ✓ FixedPointNumbers
  ✓ ColorTypes
  ✓ Colors
  4 dependencies successfully precompiled in 4 seconds. 28 already precompiled. 1 skipped during auto due to previous errors.
```

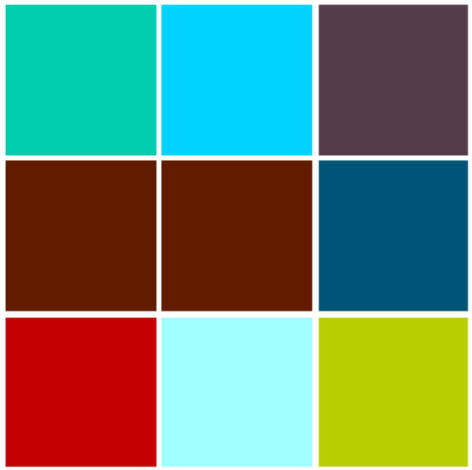
Bibliotecas

Exemplo

```
using Colors
```

```
paleta = distinguishable_colors(100)
```

```
rand(paleta, 3, 3)
```



Bibliotecas

Plot

Vamos ver agora como trabalhar com plots em Julia.
Primeiro, temos que importar o pacote “Plots"

```
using Pkg
Pkg.add("Plots")
using Plots
```

Bibliotecas

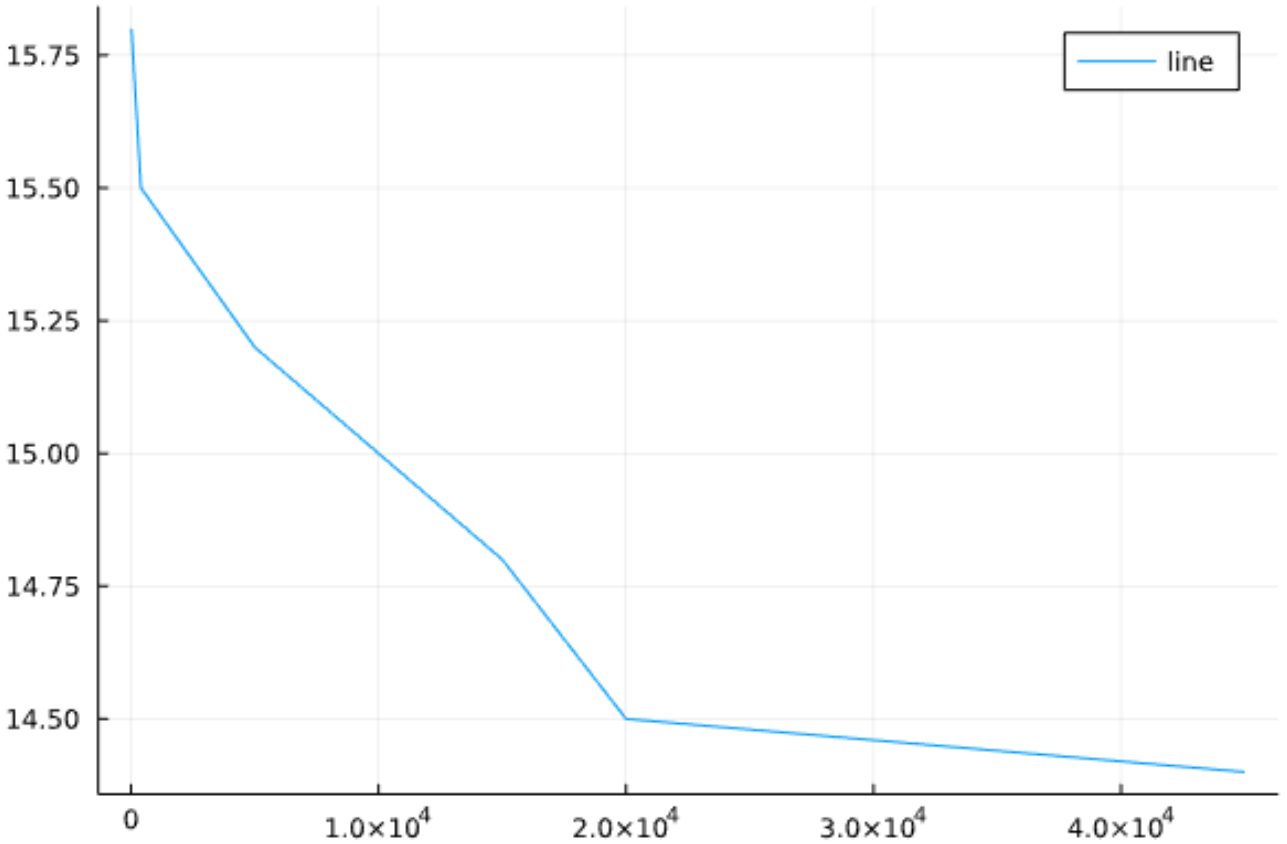
Plot

```
globaltemperatura = [14.4, 14.5, 14.8, 15.2, 15.5, 15.8]
numpiratas = [45000, 20000, 15000, 5000, 400, 17];
```

Bibliotecas

Plot

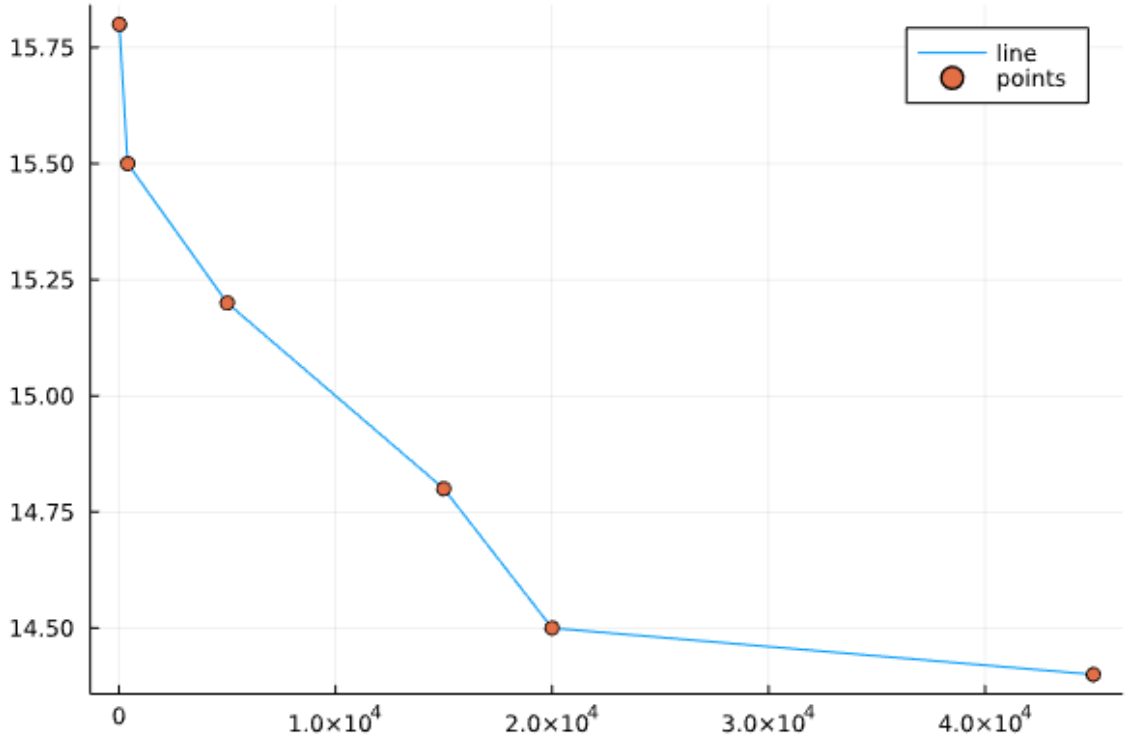
```
#Plotamos
plot(numpiratas, globaltemperatura, label="line")
```



Bibliotecas

Plot

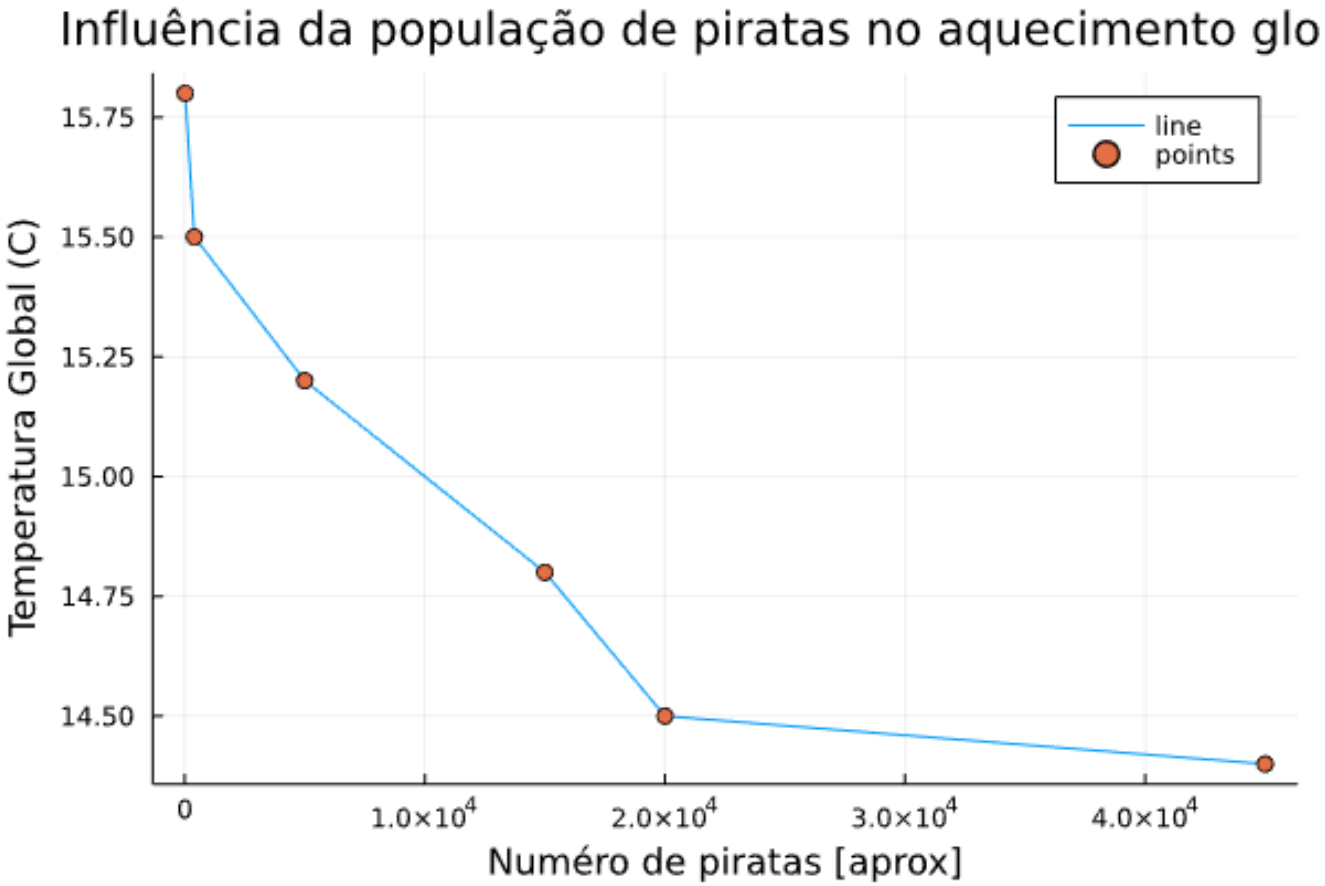
```
# Plotamos marcando os pontos
plot(numpiratas, globaltemperatura, label="line")
scatter!(numpiratas, globaltemperatura, label="points")
```



Bibliotecas

Plot

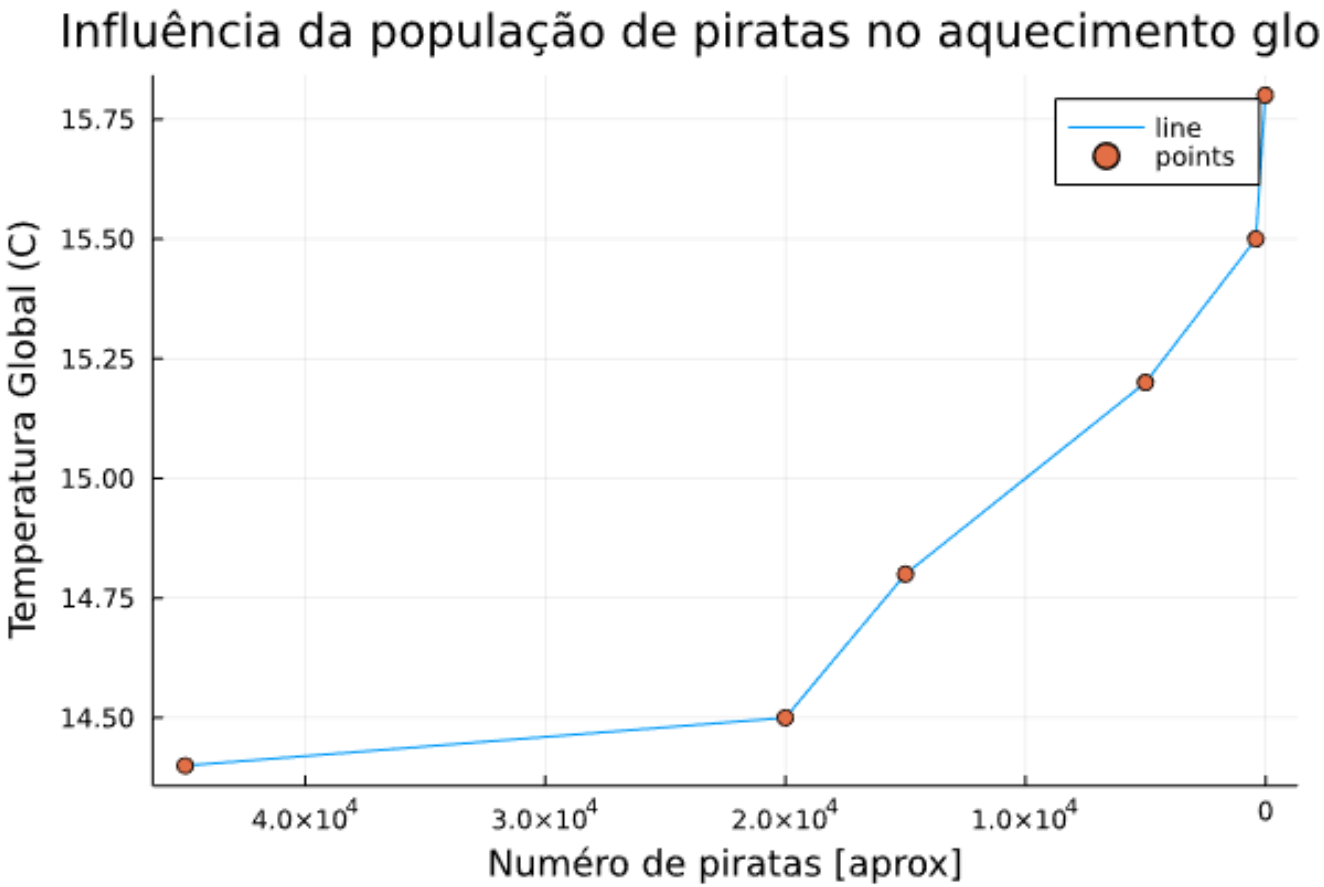
```
xlabel!("Numéro de piratas [aprox]")
ylabel!("Temperatura Global (C)")
title!("Influência da população de piratas no aquecimento global")
```



Bibliotecas

Plot

```
xflip!()
```



Exercício Geral

Crie um programa que seja capaz de cadastrar pessoas em uma loja de peças eletrônicas. A cada compra que uma pessoa faz, ela recebe o valor do produto em pontos de fidelidade. Quando a pessoa atinge 7000 pontos, ela tem direito a um desconto de 15% na próxima compra. O programa deve receber o nome da pessoa, checar se ela já é cadastrada e, se for, adicionar os pontos em seu cadastro; se não, deve criar o cadastro. O programa deve receber valores dos produtos da compra até que seja digitado o valor -1. Se a pessoa chegar na quantidade de pontos necessária para ganhar o desconto, o mesmo deve ser aplicado no valor final da compra.

Próximos passos...

- Cursos mais focados em áreas específicas podem ser achados na Julia Academy
- Treinar e escrever projetos maiores na linguagem
- Reescrever sistemas feitos em Python e Matlab em Julia

Contato

João Víctor Costa de Oliveira: joao.oliveira@ice.ufjf.br