

Processamento de Imagem

Carregando a Imagem

```
url =  
"https://uploads6.wikiart.org/images/giorgio-de-chirico/the-nostalgia-of-the-infinite-1913.jpg!Large.jpg"  
• #Imagem de uma das minhas pinturas favoritas de Giorgio de Chirico  
• url = "https://uploads6.wikiart.org/images/giorgio-de-chirico/the-nostalgia-of-the-infinite-1913.jpg!Large.jpg"  
  
"infinite.jpg"  
• download(url, "infinite.jpg")
```



```
• begin
•   using Images
•   infinite = load("infinite.jpg")
• end
```

```
Array{RGB{Normed{UInt8,8}},2}
```

```
• typeof(infinite)
```

RRBX é a representação de um pixel!



```
• RGBX(1, 1, 0.5)
```

```
(600, 267)
```

```
• size(infinite)
```

Pode-se acessar o píxel correspondente da imagem



- `infinite[100, 50]`

Cortando a Imagem

Pode-se dar slice na imagem



- `infinite[250:350, 40:140]`

Vamos capturar algo mais interessante, como as pessoas na pintura



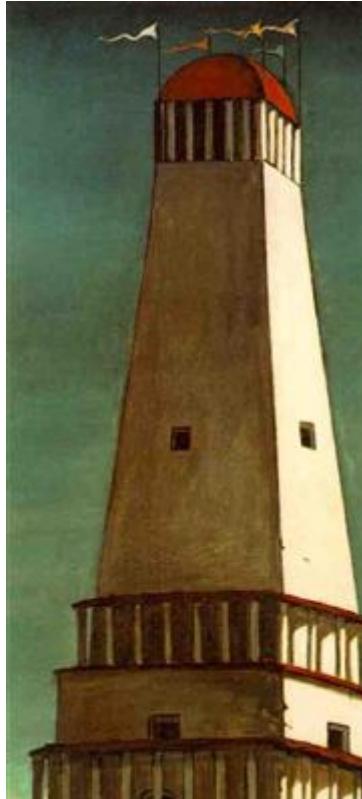
- `begin`
- `(h,l) = size(infinite)`
- `pessoas = infinite[(4*h ÷ 7):(5*h ÷ 7),(2*l÷6):(3*l÷6)]`
- `end`

pode-se usar notações de latex no Julia: \div é divisão de inteiros, escreve-se com "`\div`" + <TAB>

`(87, 45)`

- `size(pessoas)`

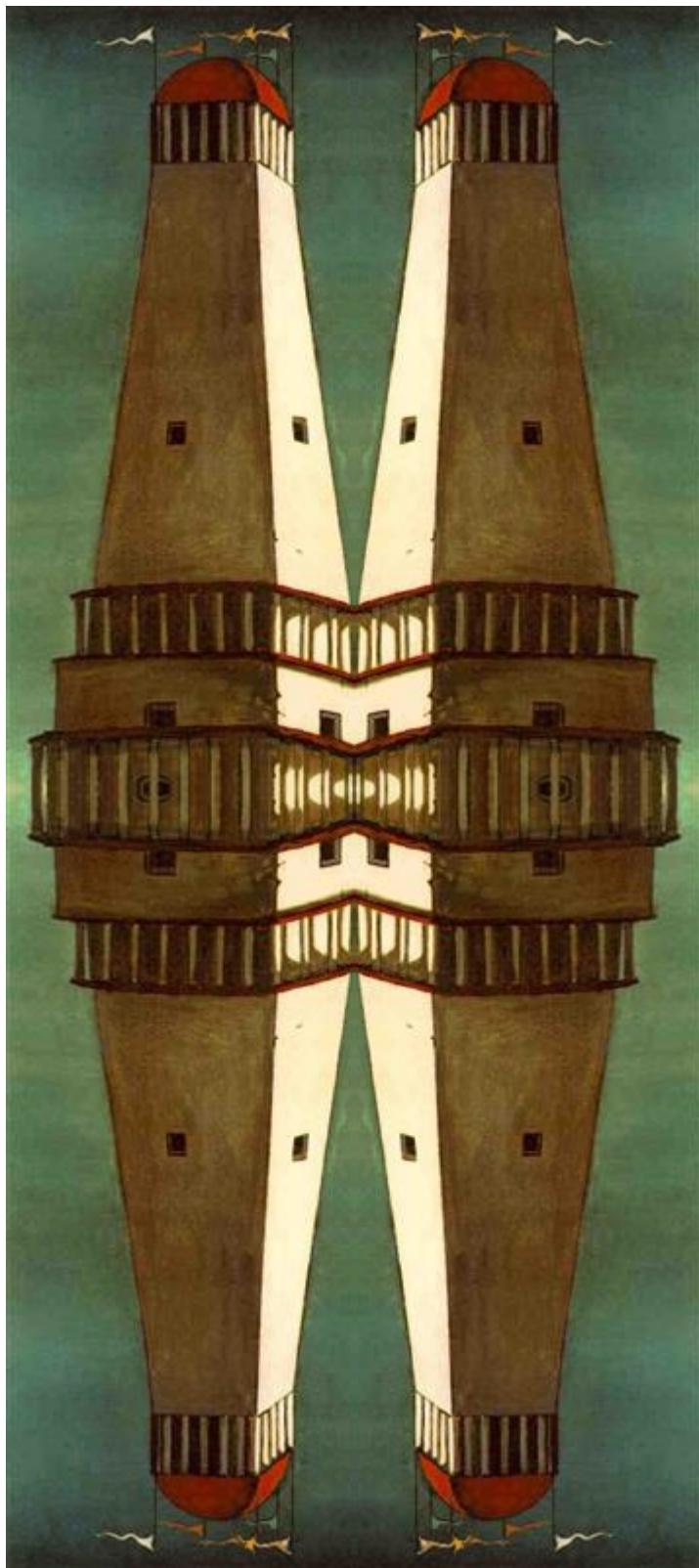
Concatenando Imagens



```
• begin
•     (h2,l2) = size(infinite)
•     predio = infinite[1:(2*h2÷3), 1:(2*l2÷3)]
• end
```



```
• [predio predio]
```



- [
- predio reverse(predio, dims=2)
- reverse(predio, dims=1) reverse(reverse(predio, dims=2), dims=1)
-]

Modificando uma Imagem

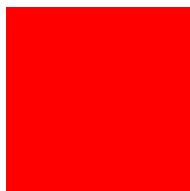
```
novo_ininite =
```



- novo_ininite = copy(infinite)

Vamos pintar a imagem copiada com alguma cor

vermelho =



- vermelho = RGB(1, 0, 0)

- for i in 1:100
- for j in 1:30
- novo_ininite[i, j] = vermelho
- end
- end



- novo_ininite

Porém, ao invés de utilizar algo como os loops, pode-se usar uma especificidade do Julia chamada de **Broadcasting**

Quando há um `.` na operação, basicamente ele está fazendo aquela operação elemento-a-elemento (*element-wise*)



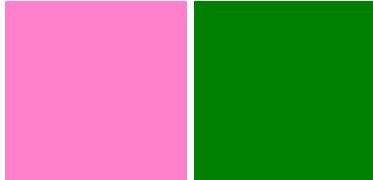
```
• begin
•     novo_ininite2 = copy(novo_ininite)
•     novo_ininite2[100:200, 1:30] .= RGB(0, 0, 1) #repares no "."
•     novo_ininite2
• end
```

Trabalhando com operações *element-wise*

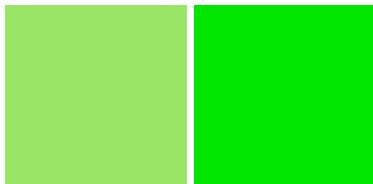
As operações *element-wise* podem ser trabalhadas com outras operações que não as definidas no próprio Julia. Para isso, vamos definir uma função, `verdificar`

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

```
• function verdificar(cor)
•     return RGB(0, cor.g, 0)
• end
```



```
• begin
•     cor = RGB(1,0.5,0.8)
•     [cor, verificar(cor)]
• end
```



```
• begin
•     cor2 = RGB(0.6,0.9,0.4)
•     [cor2, verificar(cor2)]
• end
```

type Array has no field g

1. `getproperty(::Array{ColorTypes.RGB{FixedPointNumbers.Normed{UInt8,8}},2},
 ::Symbol) @ Base.jl:33`
 2. `verificar(::Array{ColorTypes.RGB{FixedPointNumbers.Normed{UInt8,8}},2}) @ [Other]
 2`
 3. `top-level scope @ [Local: 1] [inlined]`
- `verificar(novo_ininite)`

Repare que o erro ocorreu porque a função `verificar` recebe como parâmetro apenas uma cor - um pixel - agora repare no que acontece quando utilizamos o `.` (operação *element-wise*)



- `verdificar.(novo_ininite)`

Vamos criar outra função agora chamada de `clarear`, o que ela fará é somar n aos pixeis da imagem

`clarear (generic function with 1 method)`

- `function clarear(cor, x)`
- `return RGB(cor.r + x, cor.g + x, cor.b + x)`
- `end`



- clarear.(novo_ininite2, 0.3)