

Noções de abstrações

Na célula abaixo apenas estipulamos os pacotes que utilizaremos

```
• begin
•     using Pkg
•     Pkg.add("Images")
•     using Images
• end
```

Arrays de Inteiros

Na computação *arrays* (geralmente traduzidos como vetores em português, porém vou evitar essa tradução já que *vectors* em Julia tem um significado mais específico) são estruturas extremamente úteis e relevante, talvez as mais utilizadas. Basicamente eles são arranjos de alguma tipo específico (inteiros, floats, strings, etc) onde cada um desses elementos possui um endereço - um índice.

```
elemento = 1
• elemento = 1
```

```
3x4 Array{Int64,2}:
 1  1  1  1
 1  1  1  1
 1  1  1  1
• fill(elemento, 3, 4)
```

```
Int64
• typeof(elemento)
```

Arays de Floats

É fácil notar que é simples criar um elemento de outra coisa que não inteiros, simplesmente podemos mudar o tipo do elemento. Lembre-se que *Floats* são, basicamente números reais - "números com vírgula"

```
elemento_float = 1.0
• elemento_float = 1.0
```

```
3x4 Array{Float64,2}:
 1.0  1.0  1.0  1.0
```

```
1.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0
• fill(elemento_float, 3, 4)
```

Float64

- typeof(elemento_float)

Arrays de Strings

Lembre-se que strings são basicamente textos - coisas que estão entre aspas ("")

```
elemento_string = "um"
• elemento_string = "um"
```

```
3x4 Array{String,2}:
"um"  "um"  "um"  "um"
"um"  "um"  "um"  "um"
"um"  "um"  "um"  "um"
```

- fill(elemento_string, 3, 4)

String

- typeof(elemento_string)

Arrays de Racionais

Em Julia possuímos uma estrura para tipos racionais:

```
elemento_racional = 1//1
• elemento_racional = 1//1
```

```
3x4 Array{Rational{Int64},2}:
1//1  1//1  1//1  1//1
1//1  1//1  1//1  1//1
1//1  1//1  1//1  1//1
```

- fill(elemento_racional, 3, 4)

Rational{Int64}

- typeof(elemento_racional)

Repare como o elemento é um "Racional de Inteiros", ou seja, um tipo que é feito de outros tipos, como os *arrays*!

Arrays de Arrays

Importante lembrar que na computação, sempre se tenta seguir padrões, logo, se criamos um *array* com essas estruturas de dados, é uma conclusão lógica que também podemos criar um *array* - um arranjo - com *arrays* de algum elemento

```
elemento_array = 2x2 Array{Int64,2}:
```

```
1 2  
3 4
```

- `elemento_array = [1 2 ; 3 4]`

```
3x4 Array{Array{Int64,2},2}:
```

```
[1 2; 3 4] [1 2; 3 4] [1 2; 3 4] [1 2; 3 4]  
[1 2; 3 4] [1 2; 3 4] [1 2; 3 4] [1 2; 3 4]  
[1 2; 3 4] [1 2; 3 4] [1 2; 3 4] [1 2; 3 4]
```

- `fill(elemento_array, 3, 4)`

```
Array{Int64,2}
```

- `typeof(elemento_array)`

Pode-se notar que os elementos de nosso *array* são, justamente, *arrays* de duas dimensões com elementos inteiros

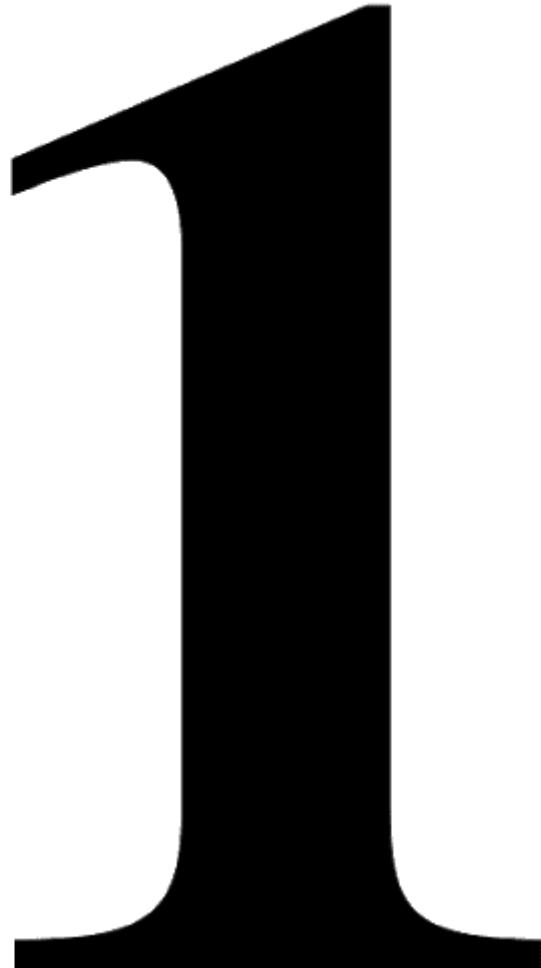
Arrays de Imagens

Logicamente, se podemos fazer um *array* de *arrays*, analogamente, podemos criar um *array* de imagens - que nada mais são do que *arrays* de pixels

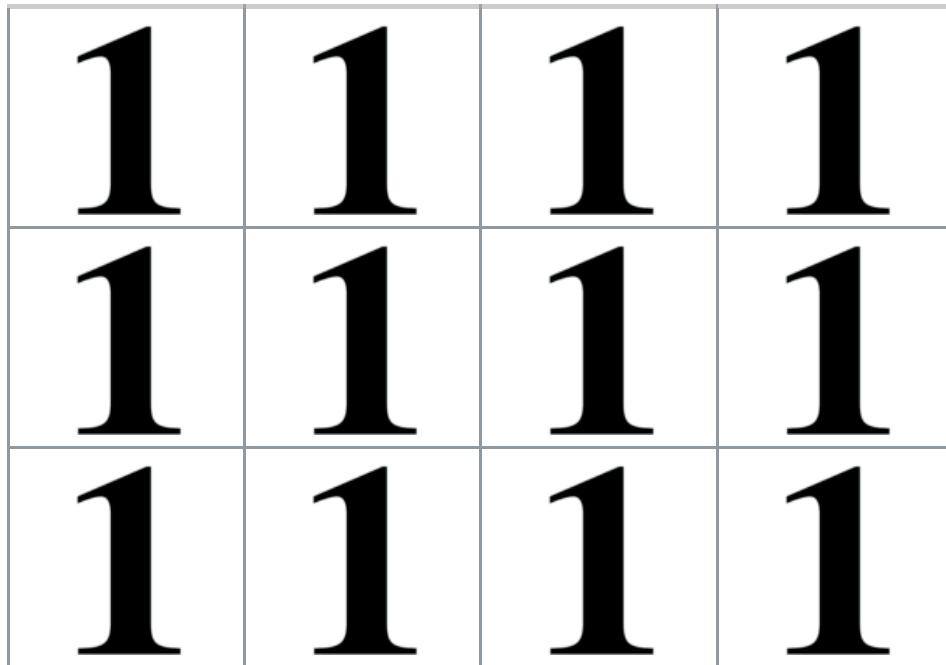
```
url =  
"https://lh3.googleusercontent.com/proxy/S_tsQgGyTd8ZpbKpGPV0RCwZhoP0rPHdxzq6h0JYkrFdchf"
```

- `url =
"https://lh3.googleusercontent.com/proxy/S_tsQgGyTd8ZpbKpGPV0RCwZhoP0rPHdxzq6h0JYkrFdchf7gSyw2cmGVMUOTb30xHt-nPH2u0DhZ2ADEXJ8dl3vGhZCa9SB-8ue7efXew7kwBW2BjMF"`

```
elemento_img =
```



```
• elemento_img = load(download(url))
```



```
• fill(elemento_img, 3, 4)
```

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

```
• typeof(elemento_img)
```

Pode-se notar como mesmo mudando os elementos, o processo da criação do *array* e como ele está representado muda pouco, ou seja, pouco importa o tipo do que é feito o *array*, ele sempre se comportará de uma maneira independente do dele. Logo escreveremos funções que manipularão *arrays* e também terão essa propriedade de se comportarem de maneira independente e previsíveis.