Presentación

Programación en Julia: Primeros pasos

Colecciones – Arreglos, matrices, vectores y tuplas

Héctor Medel

Benjamín Pérez

Matrices

- Hemos visto la sintaxis **[1, 2, 3]** crea un arreglo, en particular, un vector columna.

```
1×3 Matrix{Int64}:
 1  2  3
```

```
[1 2 3]   # Vector renglon
```

```
Int64[
    1: 1
    2: 2
    3: 3
    ]
```

```
[1, 2, 3]   # Vector columna
```

- Para crear una matriz, separamos los valores con **espacios** para las columnas, y usamos **;** para los renglones.

```
2×3 Matrix{Int64}:
 1  2  3
 4  5  6
```

```
[1 2 3; 4 5 6]
```

```
2×3 Matrix{Int64}:
 1  2  3
 4  5  6
```

```
[1 2 3;
 4 5 6]
```

- Noten que no podemos usar **,** y **;** juntas.

Productos matriciales

```
[11]
```

```
[1 2] * [3; 4]  # Vector renglon por vector columna
```

```
2×2 Matrix{Int64}:
 3  4
 6  8
```

```
[1; 2] * [3 4]  # Vector columna por vector renglon
```

```
1×2 Matrix{Int64}:
 3  8
```

```
[1 2] .* [3 4]  # Producto de vectores por elemento
```

- Ahora usaremos la función **rand()** para generar matrices cuyos elementos son números aleatorios.

```
mat1 = 20×20 Matrix{Float64}:
 0.498701   0.94968    0.292694   …  0.35911   0.316409   0.99889   0.614896
 0.707956   0.0796811  0.513906      0.529853  0.306955   0.687554  0.420302
 0.90198    0.503534   0.615485      0.816119  0.586061   0.203988  0.932868
 0.0949304  0.902818   0.247232      0.177476  0.612891   0.720069  0.478658
 0.31995    0.294427   0.042064      0.432912  0.179689   0.454278  0.816342
 0.610353   0.811673   0.875485   …  0.121348  0.113055   0.851602  0.22816
 0.117763   0.524192   0.857144      0.715538  0.557443   0.459476  0.858011
 ⋮                                ⋱
 0.15344    0.933603   0.47773       0.639578  0.968712   0.489706  0.127809
 0.803024   0.198272   0.895934   …  0.572564  0.688936   0.183809  0.67343
 0.485759   0.6392     0.494902      0.516865  0.0744313  0.313849  0.893689
 0.146965   0.760626   0.0143382     0.348372  0.19344    0.96224   0.483678
 0.778921   0.45881    0.500231      0.378564  0.745741   0.726847  0.698415
 0.489332   0.13641    0.0874629     0.563262  0.657764   0.448493  0.586307
```

```
mat1 = rand(20,20)
```

Tenemos en memoria la matriz **mat1**

```
2
```

```
ndims(mat1)
```

```
(20, 20)
```

```
nrows, ncols = size(mat1)
```

```
20
```

```
nrows
```

```
400
```

```
length(mat1)
```

```
(400)
```

```
size(mat1[:])
```

Algunas matrices que comúnmente usamos: zeros(N,M), ones(N,M), Identidad, ...

```
5×5 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
```

```
zeros(5 ,5)
```

```
5×5 Matrix{Float64}:
 1.0  1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0  1.0
```

```
ones(5 ,5)
```

La matriz identidad se encuentra dentro del paquete LinearAlgebra

```
using LinearAlgebra
```

```
rrmat = 5×5 Matrix{Float64}:
 0.204801   0.928892  0.959896  0.105952  0.244197
 0.182013   0.978682  0.83426   0.521465  0.381093
 0.0815078  0.769392  0.939435  0.737494  0.73645
 0.0140886  0.10529   0.603693  0.614724  0.587156
 0.392042   0.724491  0.90613   0.256833  0.443794
```

```
rrmat = rand(5,5)
```

```
UniformScaling{Bool}
true*I
```

```
I  # En muchas ocasiones no necesitamos generar la matriz...  podemos tener un objeto identidad con sus propiedades definidas
```

```
5×5 Matrix{Bool}:
 1  0  0  0  0
 0  1  0  0  0
 0  0  1  0  0
 0  0  0  1  0
 0  0  0  0  1
```

```
Matrix(I,5,5)   # Sin embargo, en otras ocasiones puede ser util generar una matriz...
```

```
5×5 Matrix{Float64}:
 1.2048     0.928892  0.959896  0.105952  0.244197
 0.182013   1.97868   0.83426   0.521465  0.381093
 0.0815078  0.769392  1.93943   0.737494  0.73645
 0.0140886  0.10529   0.603693  1.61472   0.587156
 0.392042   0.724491  0.90613   0.256833  1.44379
```

```
rrmat + I
```

```
5×5 Matrix{Float64}:
 1.2048     0.928892  0.959896  0.105952  0.244197
 0.182013   1.97868   0.83426   0.521465  0.381093
 0.0815078  0.769392  1.93943   0.737494  0.73645
 0.0140886  0.10529   0.603693  1.61472   0.587156
 0.392042   0.724491  0.90613   0.256833  1.44379
```

```
rrmat + Matrix(I,5,5)
```

```
[0.204801, 0.978682, 0.939435, 0.614724, 0.443794]
```

```
rrmat[Matrix(I,5,5)]   # Podemos usar la matriz identidad para accesar a los valores de la diagonal principal
```

reshape

- Nos permite cambiar las dimensiones de una matriz (si es posible).

```
vecA = Int64[
           1:  1
           2:  2
           3:  3
           4:  4
           5:  5
           6:  6
           7:  7
           8:  8
           9:  9
           10:  10
           11:  11
           12:  12
           ]
```

```
vecA = collect(1:12)
```

```
2×6 Matrix{Int64}:
 1  3  5  7   9  11
 2  4  6  8  10  12
```

```
reshape(vecA, 2, 6)
```

```
matB = 3×4 Matrix{Int64}:
       1   2   3   4
       5   6   7   8
       9  10  11  12
```

```
matB = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

```
4×3 Matrix{Int64}:
 1   6  11
 5  10   4
 9   3   8
 2   7  12
```

```
reshape(matB,4,3)
```

repeat

- Nos permite repetir arreglos para generar un arreglo de dimensión diferente.

```
matC = 2×2 Matrix{Int64}:
       1  2
       3  4
```

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

```
4×8 Matrix{Int64}:
 1  2  1  2  1  2  1  2
 3  4  3  4  3  4  3  4
 1  2  1  2  1  2  1  2
 3  4  3  4  3  4  3  4
```

```
repeat(matC, 2, 4)
```

```
12×16 Matrix{Int64}:
 1  1  2  2  1  1  2  2  1  1  2  2  1  1  2  2
 1  1  2  2  1  1  2  2  1  1  2  2  1  1  2  2
 3  3  4  4  3  3  4  4  3  3  4  4  3  3  4  4
 3  3  4  4  3  3  4  4  3  3  4  4  3  3  4  4
 1  1  2  2  1  1  2  2  1  1  2  2  1  1  2  2
 1  1  2  2  1  1  2  2  1  1  2  2  1  1  2  2
 3  3  4  4  3  3  4  4  3  3  4  4  3  3  4  4
 3  3  4  4  3  3  4  4  3  3  4  4  3  3  4  4
 1  1  2  2  1  1  2  2  1  1  2  2  1  1  2  2
 1  1  2  2  1  1  2  2  1  1  2  2  1  1  2  2
 3  3  4  4  3  3  4  4  3  3  4  4  3  3  4  4
 3  3  4  4  3  3  4  4  3  3  4  4  3  3  4  4
```

```
repeat(matC, inner=(2,2), outer=(3,4))
```

Una tupla es un grupo de valores de tamaño fijo, separado por comas y entre paréntesis.

- Puede contener distintos tipos de valores.
- Es un contenedor heterogéneo.

```
(1, 2.4, 'c', "Hola", [1, 2, 3, 4])
```

```
a1, a2, a3, a4, a5 = 1, 2.4, 'c', "Hola", collect(1:4)
```

```
t1 = (1, 2.4, 'c', "Hola", [1, 2, 3, 4])
```

```
t1 = a1,a2,a3,a4,a5
```

```
Tuple{Int64, Float64, Char, String, Vector{Int64}}
```

```
typeof(t1)
```