Introduction to Julia Programming

Collections, arrays and linear algebra

by Yueh-Hua Tu

Outline

- Collections
    - Arrays
    - Sets
    - Dictionaries
- Linear algebra

Collections

Arrays

The fundamental collection and data structure in a language.

Create an array

In [1]:
```
x = []
```

Out[1]:
```
0-element Array{Any,1}
```

Homogeneous: 同質性，Array 中只能放入屬於同一型別的物件

In [2]:
```
Any[]
```

Out[2]:
```
0-element Array{Any,1}
```

In [3]:
```
Int64[]
```

Out[3]:
```
0-element Array{Int64,1}
```

Type inference on array

In [4]:
```
x = [1, 2, 3]
```

Out[4]:
```
3-element Array{Int64,1}:
 1
 2
 3
```

In [5]:
```
x = [1, 1.2]
```

Out[5]:
```
2-element Array{Float64,1}:
 1.0
 1.2
```

Specified array type

In [6]:
```
Int8[1, 2, 3, 4]
```

Out[6]:
```
4-element Array{Int8,1}:
 1
 2
 3
 4
```

In [7]:
```
Array{Int8, 1}(undef, 5)  # uninitialized
```

Out[7]:
```
5-element Array{Int8,1}:
 -112
   -8
   86
   -1
  -27
```

Indexing

Index starts from 1.

□□□

1 2 3

In [8]:
```
x
```

Out[8]:
```
2-element Array{Float64,1}:
 1.0
 1.2
```

In [9]:
```
x[1]
```

Out[9]:
```
1.0
```

In [10]:
```
x[2]
```

Out[10]:
```
1.2
```

In [11]:
```
length(x)
```

Out[11]:
```
2
```

In [12]:
```
x = [6.0, 3.2, 7.6, 0.9, 2.3];
```

In [13]:
```
x[1:2]
```

Out[13]:
```
2-element Array{Float64,1}:
 6.0
 3.2
```

In [14]:
```
x[3:end]
```

Out[14]:
```
3-element Array{Float64,1}:
 7.6
 0.9
 2.3
```

In [15]:
```julia
x[begin:3]
```

Out[15]:
```
3-element Array{Float64,1}:
 6.0
 3.2
 7.6
```

In [16]:
```
x[1:2:end]
```

Out[16]:
```
3-element Array{Float64,1}:
 6.0
 7.6
 2.3
```

In [17]:
```
view(x, 1:3)
```

Out[17]:
```
3-element view(::Array{Float64,1}, 1:3) with eltype Float64:
 6.0
 3.2
 7.6
```

Assign value

In [18]:
```
x[2] = 7.5
```

Out[18]:
```
7.5
```

In [19]:
```
x
```

Out[19]:
```
5-element Array{Float64,1}:
 6.0
 7.5
 7.6
 0.9
 2.3
```

Useful operations

In [20]:

```
push!(x, 9.0)
```

Out[20]:

```
6-element Array{Float64,1}:
 6.0
 7.5
 7.6
 0.9
 2.3
 9.0
```

In [21]:

```
y = [10.0, 3.4]
append!(x, y)
```

Out[21]:

```
8-element Array{Float64,1}:
  6.0
  7.5
  7.6
  0.9
  2.3
  9.0
 10.0
  3.4
```

In [22]:

```
x
```

Out[22]:

```
8-element Array{Float64,1}:
  6.0
  7.5
  7.6
  0.9
  2.3
  9.0
 10.0
  3.4
```

In [23]:
```
pop!(x)
```
Out[23]:
```
3.4
```
In [24]:
```
x
```
Out[24]:
```
7-element Array{Float64,1}:
  6.0
  7.5
  7.6
  0.9
  2.3
  9.0
 10.0
```

In [25]:
```
popfirst!(x)
```
Out[25]:
```
6.0
```
In [26]:
```
x
```
Out[26]:
```
6-element Array{Float64,1}:
  7.5
  7.6
  0.9
  2.3
  9.0
 10.0
```

In [27]:

```
pushfirst!(x, 6.0)
```

Out[27]:

```
7-element Array{Float64,1}:
  6.0
  7.5
  7.6
  0.9
  2.3
  9.0
 10.0
```

Random array

In [28]:
```
x = rand(5)
```

Out[28]:
```
5-element Array{Float64,1}:
 0.09810929831480641
 0.46112452279406857
 0.9618021721401293
 0.19292860821288849
 0.41253510699062246
```

In [29]:
```
sort(x)
```

Out[29]:
```
5-element Array{Float64,1}:
 0.09810929831480641
 0.19292860821288849
 0.41253510699062246
 0.46112452279406857
 0.9618021721401293
```

In [30]:
```
x
```

Out[30]:
```
5-element Array{Float64,1}:
 0.09810929831480641
 0.46112452279406857
 0.9618021721401293
 0.19292860821288849
 0.41253510699062246
```

In [31]:
```
sort!(x)
```

Out[31]:
```
5-element Array{Float64,1}:
 0.09810929831480641
 0.19292860821288849
 0.41253510699062246
 0.46112452279406857
 0.9618021721401293
```

In [32]:
```
x
```

Out[32]:
```
5-element Array{Float64,1}:
 0.09810929831480641
 0.19292860821288849
 0.41253510699062246
 0.46112452279406857
 0.9618021721401293
```

Sorting: big to small

In [33]:

```
sort(x, rev=true)
```

Out[33]:

```
5-element Array{Float64,1}:
 0.9618021721401293
 0.46112452279406857
 0.41253510699062246
 0.19292860821288849
 0.09810929831480641
```

Sorting by absolute value

In [34]:

```
x = randn(10)
```

Out[34]:

```
10-element Array{Float64,1}:
 -0.04449731782053265
 -0.24067299751804175
  0.42097259514342966
 -1.2941611564169055
  0.9251854732954373
  3.265601659774667
  1.2862744750362891
  0.06376759068522922
 -0.6098980867393012
 -0.08696207979292975
```

In [35]:

```
sort(x, by=abs)
```

Out[35]:

```
10-element Array{Float64,1}:
 -0.04449731782053265
  0.06376759068522922
 -0.08696207979292975
 -0.24067299751804175
  0.42097259514342966
 -0.6098980867393012
  0.9251854732954373
  1.2862744750362891
 -1.2941611564169055
  3.265601659774667
```

Iteration

In [36]:

```
for i in x
   println(i)
end
```

```
-0.04449731782053265
-0.24067299751804175
0.42097259514342966
-1.2941611564169055
0.9251854732954373
3.265601659774667
1.2862744750362891
0.06376759068522922
-0.6098980867393012
-0.08696207979292975
```

請造出一個陣列，當中的數值是均勻分佈，從-345到957.6

提示：$y = \frac{x - min(x)}{max(x) - min(x)}$

請造出一個陣列，當中的數值是服從常態分佈

請造出一個陣列，當中的數值是服從常態分佈，μ=3.5，σ=2.5

提示：$y = \frac{x-\mu}{\sigma}$

Sets

Mathematical set.

In [40]:
```
x = Set([1, 2, 3, 4])
```
Out[40]:
```
Set{Int64} with 4 elements:
  4
  2
  3
  1
```
In [41]:
```
push!(x, 5)
```
Out[41]:
```
Set{Int64} with 5 elements:
  4
  2
  3
  5
  1
```
In [42]:
```
pop!(x)
```
Out[42]:
```
4
```
In [43]:
```
x
```
Out[43]:
```
Set{Int64} with 4 elements:
  2
  3
  5
  1
```

Exists

In [44]:
```
3 in x
```
Out[44]:
```
true
```
In [45]:
```
4 in x
```
Out[45]:
```
false
```

Equivalent

In [46]:
```
x == Set([3, 2, 1, 5])
```
Out[46]:
```
true
```

Iteration

In [47]:

```
for i in x
   println(i)
end
```

```
2
3
5
1
```

請告訴我以下資料有幾種數值

[8, 4, 1, 2, 9, 4, 5, 4, 5, ...]

Dictionaries

A data structure stores key-value pairs.

In [50]:
```
x = Dict("1" => 1, "2" => 2, "3" => 3)
```

Out[50]:
```
Dict{String,Int64} with 3 entries:
  "1" => 1
  "2" => 2
  "3" => 3
```

In [51]:
```
x["1"]
```

Out[51]:
```
1
```

In [52]:
```
x["A"]
```

```
KeyError: key "A" not found

Stacktrace:
 [1] getindex(::Dict{String,Int64}, ::String) at ./dict.jl:477
 [2] top-level scope at In[52]:1
```

Add new pair

In [53]:
```
x["4"] = 4
```

Out[53]:
```
4
```

In [54]:
```
x
```

Out[54]:
```
Dict{String,Int64} with 4 entries:
  "4" => 4
  "1" => 1
  "2" => 2
  "3" => 3
```

Overwrite

In [55]:
```
x["1"] = 5
```
Out[55]:
```
5
```
In [56]:
```
x
```
Out[56]:
```
Dict{String,Int64} with 4 entries:
  "4" => 4
  "1" => 5
  "2" => 2
  "3" => 3
```

keys and values

In [57]:
```
keys(x)
```

Out[57]:
```
Base.KeySet for a Dict{String,Int64} with 4 entries. Keys:
  "4"
  "1"
  "2"
  "3"
```

In [58]:
```
values(x)
```

Out[58]:
```
Base.ValueIterator for a Dict{String,Int64} with 4 entries. Values:
  4
  5
  2
  3
```

Iteration

In [59]:

```
for (k, v) in x
    println(k, "->", v)
end
```

```
4->4
1->5
2->2
3->3
```

Linear Algebra

Linear systems

Linear systems are most common in scientific computing. We have a linear system such as follow:

$$\begin{cases} w_{11}x + w_{12}y + w_{13}z = a \\ w_{21}x + w_{22}y + w_{23}z = b \\ w_{31}x + w_{32}y + w_{33}z = c \end{cases}$$

We usually express in matrix form:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

And we use symbols to represent matrices and vectors.

$$Ax = b$$

Inner product

Inner product is defined as following, and transpose is needed to calculate inner product.

$$A^T B$$

In [60]:

```
A = [1 2 3; 4 5 6; 7 8 9]
B = [12 11 10 9; 8 7 6 5; 4 3 2 1]
A' * B
```

Out[60]:

```
3×4 Array{Int64,2}:
  72   60  48  36
  96   81  66  51
 120  102  84  66
```

Determinant

Determinant (行列式) is calculated by `det`. Here we need built-in LinearAlgebra package.

In [61]:
```
using LinearAlgebra
```

In [62]:
```
det(A)
```

Out[62]:
```
6.661338147750939e-16
```

Rank

Rank (秩) is calculated by `rank`. LinearAlgebra package is needed.

In [63]:
```
rank(A)
```

Out[63]:
```
2
```

Trace

Trace is calculated by `tr`. LinearAlgebra package is needed.

In [64]:
```
tr(A)
```

Out[64]:
```
15
```

Norm

Norm is used to calculate the "length" of a vector. Usually, we use Euclidean norm (歐幾里得範數). LinearAlgebra package is needed.

$$||v||_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

In [65]:

```
b = [5, 5, 5]
norm(b)
```

Out[65]:

```
8.660254037844387
```

If a norm is calculated over a matrix, Frobenius norm is calculated. LinearAlgebra package is needed.

In [66]:

```
norm(A)
```

Out[66]:

```
16.881943016134134
```

p-norm

P-norm is the norm which is generalized to the power of p .

$$||v||_p = (\sum_i |v_i|^p)^{\frac{1}{p}}$$

In [67]:
```
norm(b, 5)
```

Out[67]: 6.228654698077587

A Manhattan Distance (or Taxicab norm, l_1 norm) is often used in machine learning models.

$$||v||_1 = |v_1| + |v_2| + \cdots + |v_n|$$

In [68]:
```
norm(b, 1)
```

Out[68]: 15.0

Infinity norm

$$||v||_{\infty} = max(|v_1|, |v_2|, \dots, |v_n|)$$

In [69]:
```
norm(b, Inf)
```

Out[69]:
```
5.0
```

Identity matrix

Identity matrix is really easy to use in Julia, unlike `eye` in other languages. One can represent 3x3 identity matrix as follow:

In [70]:
```
Matrix{Float64}(I, 3, 3)
```

Out[70]:
```
3×3 Array{Float64,2}:
 1.0  0.0  0.0
 0.0  1.0  0.0
 0.0  0.0  1.0
```

A more convenient way is to use `I` to represent identity. The dimension of an identity matrix is determined automatically.

In [71]:
```
A * I
```

Out[71]:
```
3×3 Array{Int64,2}:
 1  2  3
 4  5  6
 7  8  9
```

Inverse

An inverse matrix (反矩陣) is calculated by `inv` .

In [72]:
```
inv(A)
```

Out[72]:
```
3×3 Array{Float64,2}:
 -4.5036e15   9.0072e15   -4.5036e15
  9.0072e15  -1.80144e16   9.0072e15
 -4.5036e15   9.0072e15   -4.5036e15
```

If a matrix is not a square and full-rank, Moore-Penrose pseudo-inverse is calculated by `pinv` .

In [73]:
```
pinv(B)
```

Out[73]:
```
4×3 Array{Float64,2}:
 -0.1125      0.1         0.3125
 -0.0166667   0.0333333   0.0833333
  0.0791667  -0.0333333  -0.145833
  0.175      -0.1        -0.375
```

Solve linear systems

To solve a linear system, we express a linear system as follow.

$$Ax = b$$

If matrix A is **invertible (可逆的)**, $\backslash$ operation is used to solve the system.

$$x = A^{-1}b$$

In [74]:
```
x = A \ b
```

Out[74]:
```
3-element Array{Float64,1}:
 -8.999999999999998
 12.999999999999998
 -4.0
```

Eigenvalue decomposition

Eigenvalue decomposition (特徵值分解) is very important and widely used. eigen returns the **eigenvalues (特徵值)** and **eigenvectors (特徵向量)** of a matrix.

In [75]:

```
vals, vecs = eigen(A)
```

Out[75]:

```
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
values:
3-element Array{Float64,1}:
 -1.1168439698070427
 -1.3036777264747022e-15
 16.116843969807043
vectors:
3×3 Array{Float64,2}:
 -0.78583    0.408248  -0.231971
 -0.0867513  -0.816497  -0.525322
  0.612328    0.408248  -0.818673
```

Eigenvalue decomposition

eigvals returns eigenvalues only.

In [76]:
```
eigvals(A)
```

Out[76]:
```
3-element Array{Float64,1}:
 -1.1168439698070427
 -1.3036777264747022e-15
 16.116843969807043
```

eigvecs returns only eigenvectors, which are in column-wise manner.

In [77]:
```
eigvecs(A)
```

Out[77]:
```
3×3 Array{Float64,2}:
 -0.78583    0.408248  -0.231971
 -0.0867513  -0.816497  -0.525322
  0.612328   0.408248  -0.818673
```

Eigenvalue decomposition

eigmax returns maximum of eigenvalues.

In [78]:
```
eigmax(A)
```

Out[78]:
```
16.116843969807043
```

eigmin returns minimum of eigenvalues.

In [79]:
```
eigmin(A)
```

Out[79]:
```
-1.1168439698070427
```

Singular value decomposition

Singular value decomposition (奇異值分解) is also a widely used matrix decomposition method. svd is used to get the results.

In [80]:
```
U, Σ, V = svd(A)
```

Out[80]:
```
SVD{Float64,Float64,Array{Float64,2}}
U factor:
3×3 Array{Float64,2}:
 -0.214837   0.887231   0.408248
 -0.520587   0.249644  -0.816497
 -0.826338  -0.387943   0.408248
singular values:
3-element Array{Float64,1}:
 16.84810335261421
  1.0683695145547103
  1.472808250397788e-16
Vt factor:
3×3 Array{Float64,2}:
 -0.479671  -0.572368   -0.665064
 -0.776691  -0.0756865   0.625318
  0.408248  -0.816497    0.408248
```

Singular value decomposition

svdvals returns **singular values (奇異值)** in descending order.

In [81]:

```
svdvals(A)
```

Out[81]:

```
3-element Array{Float64,1}:
 16.84810335261421
  1.0683695145547103
  1.4728082503977878e-16
```

Pearson correlation coefficient

In [82]:
```
using Statistics
A = [1.2 5.21 5.8;
    1.3 8.6 7.4;
    12. 6. 3.]
b = [5.2, 4.6, 7.2]
```

Out[82]:
```
3-element Array{Float64,1}:
 5.2
 4.6
 7.2
```

In [83]:
```
cor(b)
```

Out[83]:
```
1.0
```

In [84]:
```
cor(A)
```

Out[84]:
```
3×3 Array{Float64,2}:
  1.0       -0.286874  -0.930333
 -0.286874   1.0        0.618192
 -0.930333   0.618192   1.0
```

Pearson correlation coefficient

In [85]:

```
cor(A, b)
```

Out[85]:

```
3×1 Array{Float64,2}:
  0.973610396858362
 -0.4979278975048893
 -0.9894723165936354
```

In [86]:

```
B = [1.3 5.6 5.7 4.1;
    5.4 2.3 4.5 7.7;
    1.2 1.4 8.7 9.5]
cor(A, B)
```

Out[86]:

```
3×4 Array{Float64,2}:
 -0.511052  -0.671761   0.958503   0.761178
  0.970029  -0.516922  -0.548068   0.402921
  0.79066    0.353308  -0.996271  -0.470317
```

Q & A