Módulo VII: Estructuras de datos

- Celdas.
- Mapas.
- Estructuras.
- Funciones.

Celdas

Agrupación de objetos que pueden ser de distintas clases:

```
% Almacenar en una celda un número, un string, un vector numérico
% y un vector lógico:
Cell_1 = {8, 'hello', [14, 3,0], logical([1,0])}
```

```
Cell_1 = 1×4 cell array
    {[8]}    {'hello'}    {1×3 double}    {1×2 logical}
```

```
Cell_1{3} % Indexación
```

```
ans = 1×3
    14     3     0
```

```
[ A, B, C ] = Cell_1{2:end} % Indexar con rango
```

```
A =
'hello'
B = 1×3
    14     3     0
C = 1×2 logical array
   1   0
```

```
Cell_1{1} = [8, 3, 1] % Reemplazar
```

```
Cell_1 = 1×4 cell array
    {1×3 double}    {'hello'}    {1×3 double}    {1×2 logical}
```

```
% Agregar un elemento nuevo a la celda. Nótese que
% la celda originalmente tiene 4 posiciones. Si se
% agrega la nueva en la posición 6, se genera también
% la posición 5 con un double 0x0
```

```
Cell_1{6} = 'haha'
```

```
Cell_1 = 1×6 cell array
    {1×3 double}    {'hello'}    {1×3 double}    {1×2 logical}    {0×0 double}    {'haha'}
```

Mapas:

Colecciones indexadas a través de una clave, en lugar de un número.

```
fruits = {'apple', 'orange', 'cherry', 'lemon'}
```

```
fruits = 1×4 cell array
    {'apple'}    {'orange'}    {'cherry'}    {'lemon'}
```

```
amount = [10, 5, 2, 3];
```

```
% Se genera un mapa a partir de unas claves (celda) y
% unos valores asociados (vector numérico).
M = containers.Map(fruits, amount)
```

```
M =
  Map with properties:

        Count: 4
      KeyType: char
    ValueType: double
```

```
manzanas = M('apple')
```

```
manzanas = 10
```

```
M.keys
```

```
ans = 1×4 cell array
    {'apple'}    {'cherry'}    {'lemon'}    {'orange'}
```

```
M.values
```

```
ans = 1×4 cell array
    {[10]}    {[2]}    {[3]}    {[5]}
```

```
% Se puede generar también con una celda como values:
in_fridge = {[1, 0, 2], [5, 7, 8], [0, 1, 0], [9, 8, 6]}
```

```
in_fridge = 1×4 cell array
    {1×3 double}    {1×3 double}    {1×3 double}    {1×3 double}
```

```
M_2 = containers.Map(fruits, in_fridge)
```

```
M_2 =
  Map with properties:

        Count: 4
      KeyType: char
    ValueType: any
```

```
M_2.keys
```

```
ans = 1×4 cell array
    {'apple'}    {'cherry'}    {'lemon'}    {'orange'}
```

```
M_2.values
```

```
ans = 1×4 cell array
    {1×3 double}    {1×3 double}    {1×3 double}    {1×3 double}
```

Estructuras

Objetos con diferentes campos para almacenar variables de todo tipo:

```
student_1 = struct(); % Declarar la estructura
student_1.name = 'Cristian' % Crear un campo de nombre
```

```
student_1 = struct with fields:
    name: 'Cristian'
```

```
student_1.id = 5555555 % Crear un campo de id
```

```
student_1 = struct with fields:
    name: 'Cristian'
      id: 5555555
```

```
student_1.email = 'mi@yahoo.123' % Crear un campo de email
```

```
student_1 = struct with fields:
     name: 'Cristian'
       id: 5555555
    email: 'mi@yahoo.123'
```

```
student_1.name % Acceder al valor almacenado en cierto campo
```

```
ans =
'Cristian'
```

Funciones

Bloques de código invocados a través de un nombre y que cumplen cierta 'firma' o estructura de parámetros requeridos y variables retornadas como respuesta.

Ex. 1. Hacer una función que sume los dígitos de un número.

```
num_2_test = 570;
suma = count_digits(num_2_test)
```

```
suma = 12
```

Ex. 2. Haga una función que, dado un vector, encuentre el promedio de todos sus valores desde el primero hasta la posición actualmente analizada.

```
P = [1, 0, -3, 4, 7, 6, 5, 2, 2, 4];
U = get_the_means(P)
```

```
S = 1×10
    1.0000    0.5000   -0.6667    0.5000    1.8000    2.5000    2.8571    2.7500 ···
```

Ex. 3. Haga una función que reciba como argumento de entrada una palabra y como argumento de salida entregue un mapa, donde sus claves sean las letras diferentes que hay en la palabra y sus valores sean la cantidad de veces que se repiten las letras.

```
palabra = 'esternocleidomastoideo';
out_map = build_dictionary( palabra );
```

```
cell_letters = 1×12 cell array
    {'a'}    {'c'}    {'d'}    {'e'}    {'i'}    {'l'}    {'m'}    {'n'}    {'o'}    {'r'}    {'s'}    {'
count_letters = 12×1
     1
     1
     2
     4
     2
     1
     1
     1
     4
```

```
1
⋮
```

```
out_map.keys
```

```
ans = 1×12 cell array
    {'a'}    {'c'}    {'d'}    {'e'}    {'i'}    {'l'}    {'m'}    {'n'}    {'o'}    {'r'}    {'s'}    {'
```

```
out_map.values
```

```
ans = 1×12 cell array
    {[1]}    {[1]}    {[2]}    {[4]}    {[2]}    {[1]}    {[1]}    {[1]}    {[4]}    {[1]}    {[2]}    {[
```

Ex. 4. Haga una función que dado un arreglo cualquiera, mueva todos los ceros hasta el final, conservando el orden relativo de los demás valores almacenados.

```
A = randi([-5, 5], 1, 100);
A = move_zeros(A)
```

```
A = 1×100
    -2     2     2    -4    -4     5    -2     1    -3     3    -3     2     4 ···
A = 1×100
    -2     2     2    -4    -4     5    -2     1    -3     3    -3     2     4 ···
```

Ex. 5. Dadas dos matrices y dos valores a y b tales que $a < b$. Haga una función que retorne un arreglo lógico, donde el valor de cada posición sea 1 sólo si en esa posición, para ambas matrices, el valor almacenado está en el rango $[a, b]$, o 0 en caso contrario. Además, retorne estos valores que cumplen con la condición.

```
M1 = randi([-10, 10], 50, 50);
M2 = randi([-10, 10], 50, 50);
v_range = [-3, 3];
[U, ms] = get_matrix_values(M1, M2, v_range);
```

```
function [U, ms] = get_matrix_values(M1, M2, v_range)
    a = v_range(1);
    b = v_range(2);
    U = (M1 >= a) & (M1 <= b) & (M2 >= a) & (M2 <= b);

    m1_vals = M1(U);
    m2_vals = M2(U);

    ms = [m1_vals, m2_vals];
end
```

```
function A = move_zeros(A)
    N = length(A);

    for (i = 1 : N)
        if ( A(i) == 0 )
            for (j = i : N-1)
                A(j) = A(j+1);
            end
            A(end) = 0;
        end
    end
    A
end

function out_map = build_dictionary( a_word )

    unique_letters = unique(a_word); % Obtener las letras únicas
    count_letters = zeros( length(unique_letters), 1 );
    cell_letters = {}; % aquí se almacena cada letra

    for i = 1 : length(unique_letters)
        % buscar cada letra única
        index_letter = find( a_word == unique_letters(i) );
        % index_letter dice en que posiciones se encuentra la letra
        count_letters(i) = length(index_letter);
        % el tamaño de index_letter es a la vez la cantidad
        % de apariciones de la letra en la palabra
        cell_letters{i} = unique_letters(i);
        % agregar la letra al diccionario
    end

    out_map = containers.Map( cell_letters, count_letters);
end

function suma = count_digits(num_1)
    num_1_w = num2str(num_1); % Transforme el número en palabra
    N = length(num_1_w);

    suma = 0;
    for (i = 1 : N)
        % Sume cada dígito de la palabra
        suma = suma + str2num(num_1_w(i));
    end
    suma;
end

function S = get_the_means(P)
    N = length(P);
    S = [];
    for (j = 1 : N)
```

```
        % Encuentre los promedios de P desde 1 hasta
        % cada posición
        S = [S, mean( P(1:j) )];
    end
    S
end
```