Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

1.5. Funciones

Python. Definición de funciones:

def nombreFuncion(parametros)

sentencias

return [valor1], [valor2]

Invocar funciones: *nombreFuncion(argumentos)*

Instrucciones: Funciones Python

- **En Matlab las funciones se definen:**

function nombreFunción (parámetros)

sentencias

end

Ejercicio 48_01. Funciones en Python

```
# Definición de función:
# Función suma
def sumar(x,y):
  return x + y
# Llamada a una función:
# Probamos la función suma
resultado = sumar(1,3)
print("El resultado de sumar 1 y 3 es: %d" %resultado)
```

```
El resultado de sumar 1 y 3 es: 4
```

```
# Funciones sin return:
def mensaje(valor, texto):
      if(valor == 1):
            print("Mensaje con valor 1: ", texto)
            return # Cortamos la ejecución si valor = 1
      print("Mensaje con otro valor: ", texto)

 mensaje(1, "El valor es 1.")
 mensaje(2, "El valor es 2.")
```

```
Mensaje con valor 1: El valor es 1.
Mensaje con otro valor: El valor es 2.
```

```
# Lista como parámetro de una función:
# Creamos las listas
num1 = [3, 4, 5, 4]
num2 = [1, 2, 3]
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

```
# Definimos la función
def cuadrados(x):
   for valor in x:
        print(valor**2)

# Llamamos a la función
cuadrados(num1)
cuadrados(num2)
```

```
    9
    16
    25
    16

     1
     4
     9
```

```
# Funciones con parámetros por defecto y dos parámetros de salida:
# Función que calcula el área y volumen de una esfera
# Definimos la función con el valor de pi por defecto
def calcularAreaVol(r, pi=3.14):
        a = 4*pi*r**2
        v = 4/3 * pi*r**3 # Es el operador para elevar ^
        return a, v

# Llamamos a la función pasándole sólo el radio
area, vol = calcularAreaVol(2)
print("El área es %f y el volumen %f" %(area, vol))
```

```
El área es 50.240000 y el volumen 33.493333
```

```
# Si recibimos varios parámetros de salida en una sola variable, tendremos una tupla:
c = calcularAreaVol(2)
print(c)
```

```
(50.24, 33.4933333333333)
```

Ejercicio 48_02. Funciones en MATLAB

```
% Definición de función:
% Función suma hay que crearla en otro script y guardarla exactamente con el mismo
nombre (sumar.m) en la misma carpeta donde queremos trabajar y llamarla para usarla.
function resultado = sumar(x, y)
    resultado = x + y;
end

% Llamada a una función:
% Probamos la función sumar de x = 1 e y = 3
resultado = sumar(1,3);
fprintf('El resultado de sumar 1 y 3 es: %d\n.', resultado);
```

```
El resultado de sumar 1 y 3 es: 4.
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

```
% Funciones sin return:
% Si el valor es 1, mostrar un mensaje de lo contrario, mostrar otro mensaje.
% Primero hay que crear esta función mensaje en otro script y guardarla con el mismo
nombre (mensaje.m) en la misma carpeta donde queremos llamarla para usarla.

function mensaje(valor, texto)
    if(valor == 1)
        fprintf('Mensaje con valor 1: %s\n.', texto);
        return; % Cortamos la ejecución si valor = 1
    end
    fprintf('Mensaje con otro valor: %s\n.', texto);
end

% Para usarla, por ejemplo:
mensaje(1, 'El valor es 1');
mensaje(2, 'El valor es 2');

Mensaje con valor 1: El valor es 1.
Mensaje con otro valor: El valor es 2.

% Lista como parámetro de una función:
% Creamos las listas
num1 = [3, 4, 5, 4];
num2 = [1, 2, 3];

% Definimos la función
function cuadrados(x)
    for i = 1:length(x)
        disp(x(i)^2);
    end
end

% Llamamos a la función
cuadrados(num1);
cuadrados(num2);

    9
    16
    25
    16

     1
     4
     9

% Funciones con parámetros por defecto y dos parámetros de salida:
% Función que calcula el área y volumen de una esfera
% Definimos la función con el valor de pi por defecto
function [a, v] = calcularAreaVol(r, pi)
    if nargin < 2
        pi = 3.14;
    end
    a = 4*pi*r^2;
    v = (4/3) * pi * r^3;
end
```

```
% Llamamos a la función pasándole sólo el radio
[area, vol] = calcularAreaVol(2);
fprintf('El área es %f y el volumen %f\n', area, vol);

El área es 50.240000 y el volumen 33.493333

% Si recibimos varios parámetros de salida en una sola variable, tendremos una
matriz:
c = calcularAreaVol(2);
disp(c);

50.2400
```

1.5.1. Librerías e instalación

- **Cargar librerías en Python:**

  SpiPy, NumPy, matplotlib, pandas, scikit-learn

  - **NumPy**
    - Es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente indicada para grandes volúmenes de datos.
    - Permite la creación de *arrays* y matrices, eficientes de definir y manipular.
    - Un *array*, en Python, es una estructura de datos de un mismo tipo, organizada en forma de tabla o cuadrícula de distintas dimensiones.
    - Los elementos de los *arrays* deben ser del mismo tipo.
    - 1D *array* (1 dimensión), 2D *array* (2 dimensiones), 3D *array* (a dimensiones).

Documentación general sobre numpy: Librería Python numpy

Documentación *arrays* y matrices con numpy: Arrays en numpy

```
# scipy
import scipy
print('Versión de scipy: %s' % scipy._version_))
# numpy
import numpy
print('Versión de numpy: %s' % numpy._version_))
# matplotlib
import matplotlib
print('Versión de matplotlib: %s' % matplotlib._version_))
# pandas
import pandas
print('Versión de pandas: %s' % pandas._version_))
# scikit-learn
import scikit-learn
print('Versión de scikit-learn: %s' % sklearn._version_))
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

```
Versión de scipy: 1.7.1
Versión de numpy: 1.20.3
Versión de matplotlib: 3.4.3
Versión de pandas: 1.3.4
Versión de scikit-learn: 0.24.2
```

- En **MATLAB** no se cargan librerías sino Toolboxes durante la instalación del programa que hacen funciones parecidas a las librerías de Python, siendo en casos específicos más potentes. Entre estos Toolboxes destacan:
-
    - Inteligencia artificial, data analytics y estadística
        - Deep Learning Toolbox
        - Statistics and Machine Learning Toolbox
        - Curve Fitting Toolbox
        - Text Analytics Toolbox
    - Matemáticas y Optimización
        - Optimization Toolbox
        - Global Optimization Toolbox
        - Symbolic Math Toolbox
        - Mapping Toolbox
        - Partial Differential Equation Toolbox
    - Generación de informes y acceso a bases de datos
        - Database Toolbox

Documentación sobre Toolbox: MATLAB Toolbox

1.5.2. Funciones para *arrays* y vectores

Ejercicio 49_01. *Arrays* con numpy en Python

```
# Creamos un array a partir de la lista o tupla:
# import mumpy as np
a1 = [1, 2, 3] # Definimos la lista o vector
a2 = np.array(a1) # Convertimos
print("Vector original: ", a1)
print("Vector convertido a numpy: ", a2)

Vector original: [1 ,2, 3]
Vector convertido a numpy: [1 2 3]

# Podemos crear arrays directamente de 1, 2 o 3 dimensiones:
a1 = np.array([1, 2, 3, 4]) # vector o array de una dimensión
a2 = np.array([1, 2, 3],[4, 5, 6], [7, 8, 9]) # Matriz
a3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]]) # Cubo 3D
print("Vector: ", a1)
print("Matriz: ", a2)
print("Cubo: ", a3)
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

```
Vector: [1 2 3 4]
Matriz: [[1 2 3] [4 5 6] [7 8 9]]
Cubo: [[[1 2] [3 4]] [[5 6] [7 8]]]
```

Ejercicio 49_02. Vectores en MATLAB

```
% Creamos un cell array a partir de un vector
a1 = [1, 2, 3]; % Definimos el vector

% Convertimos el vector a cell array
a2 = num2cell(a1);

% Mostramos el vector original y el cell array resultante
fprintf('Vector original: ');
disp(a1);
fprintf('Cell array convertido en MATLAB: ');
disp(a2);
```

```
Vector original:      1     2     3
Cell array convertido en MATLAB: {[1]}    {[2]}    {[3]}
```

```
% Crear arrays directamente de 1, 2 o 3 dimensiones
% Vector o array de una dimensión
a1 = [1, 2, 3, 4];
fprintf('Vector: ');
disp(a1);

% Matriz
a2 = [1, 2, 3; 4, 5, 6; 7, 8, 9];
fprintf('Matriz:\n');
disp(a2);

% Cubo 3D
a3 = cat(3, [1, 2; 3, 4], [5, 6; 7, 8]);

% o bien
a3(:,:,1) = [1, 2; 3, 4];
a3(:,:,2) = [5, 6; 7, 8];
fprintf('Cubo:\n');
disp(a3);
```

```
Vector:    1     2     3     4
Matriz:
     1     2     3
     4     5     6
     7     8     9
Cubo:
(:,:,1) =
     1     2
     3     4
(:,:,2) =
     5     6
     7     8
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz

Ejercicio 50_01. Definir el tipo de datos en *arrays* con numpy en Python

```
a = np.array([1, 2, 3, 4, 5], dtype = int) # entero
print(a)
print(a.dtype) # Muestra el tipo
b = np.array([1.5, 2, 3.9], dtype = float) # float
print(b)
print(b.dtype) # Muestra el tipo
c = np.array([1, 0, 0], dtype = bool) # booleano
print(c)
print(c.dtype) # Muestra el tipo
d = np.array([1, 2, 3, 4, 5], dtype = np.int32) # Entero
print(d)
print(d.dtype) # Muestra el tipo
```

```
   [1 2 3 4 5]
int32
    [1.5 2.0 3.90]
float64
   [ True False False]
bool
   [1 2 3 4 5]
int32
```

Ejercicio 50_02. Definir el tipo de datos en vectores en MATLAB

```
% Entero
a = int32([1, 2, 3, 4, 5]);
disp('a:');
disp(a);
disp('Tipo de a:');
disp(class(a));

% Float
b = [1.5, 2, 3.9];
disp('b:');
disp(b);
disp('Tipo de b:');
disp(class(b));

% Booleano
c = logical([1, 0, 0]);
disp('c:');
disp(c);
disp('Tipo de c:');
disp(class(c));

% Entero de 32 bits
d = int32([1, 2, 3, 4, 5]);
disp('d:');
disp(d);
disp('Tipo de d:');
disp(class(d));
```

```
a:
   1   2   3   4   5
Tipo de a:
int32
b:
    1.5000    2.0000    3.9000
Tipo de b:
double
c:
   1   0   0
Tipo de c:
logical
d:
   1   2   3   4   5
Tipo de d:
int32
```

1.5.3. Funciones para *arrays* y matrices

Ejercicio 51_01. Creación de matrices con numpy en Python

```
# Matrices con numpy
import numpy as np
A = [[1, 2, 3], [3, 4, 5]] # Definimos la matriz
print("Matriz original: ", A)

# Convertimos a numpy
B = np.array(A)
print("Matriz convertida a numpy: ", B)

%% Creación directa
C = np.array([[0, 1, 2], [-1, 2, 3]], dtype = int)
print("Matriz de forma directa: ", C)
```

```
Matriz original: [[1, 2, 3], [3, 4, 5]]
Matriz convertida a numpy: [[1 2 3] [3 4 5]]
Matriz de forma directa: [[0  1  2][-1  2  3]]
```

Ejercicio 51_02. Creación de matrices

```
% Definir una matriz
A = [1, 2, 3; 3, 4, 5];
fprintf('Matriz original:\n');
disp(A);

% Convertir a MATLAB (no se necesita una conversión explícita)
B = A;
fprintf('Matriz convertida a MATLAB:\n');
disp(B);

% Crear una matriz directamente
C = int32([0, 1, 2; -1, 2, 3]);
fprintf('Matriz de forma directa:\n');
disp(C);
```

```
Matriz original:
     1     2     3
     3     4     5
Matriz convertida a MATLAB:
     1     2     3
     3     4     5
Matriz de forma directa:
    0    1    2
   -1    2    3
```

Ejercicio 52_01. Atributos de una matriz en Python

```python
# Atributos de un array
print("Matriz B: ", B)
print("Dimensiones de la matriz B: ", B.shape)
print("Número de dimensiones de la matriz B: ", B.ndim)
print("Número de elementos de la matriz B: ", B.size)
```

```
Matriz B: [[1 2 3][3 4 5]]
Dimensiones de la matriz B: (2, 3)
Número de dimensiones de la matriz B: 2
Número de elementos de la matriz B: 6
Matriz B:
     1     2     3
     3     4     5
```

Ejercicio 52_02. Atributos de una matriz en MATLAB

```
% Atributos de una matriz
fprintf('Matriz B:\n');
disp(B);
fprintf('Dimensiones de la matriz B: %s\n', mat2str(size(B)));
fprintf('Número de dimensiones de la matriz B: %d\n', ndims(B));
fprintf('Número de elementos de la matriz B: %d\n', numel(B));
```

```
Matriz B:
     1     2     3
     3     4     5

Dimensiones de la matriz B: [2 3]
Número de dimensiones de la matriz B: 2
Número de elementos de la matriz B: 6
Matriz B:
     1     2     3
     3     4     5
```

Ejercicio 53_01. Filas y columnas de una matriz en Python

```python
# filas y columnas de la matriz
print("Matriz B: ", B)
print("Primera fila de la matriz B: ", B[0])
print("Primera fila de otra forma de la matriz B: ", B[0,:])
```

```
print("Última fila de la matriz B: ", B[-1])
print("Segunda columna de la matriz B: ", B[:,1])
print("Última columna de la matriz B: ", B[:,-1])

Matriz B: [[1 2 3] [3 4 5]]
Primera fila de la matriz B:
     [1  2  3]
Primera fila otra forma: [1  2  3]
Última fila de la matriz B: [3  4  5]
Segunda columna de la matriz B: [2  4]
Última columna de la matriz B: [3  5]
```

Ejercicio 53_02. Filas y columnas de una matriz en MATLAB

```
% Filas y columnas de la matriz
fprintf('Matriz B:\n');
disp(B);
fprintf('Primera fila de la matriz B:\n');
disp(B(1, :));
fprintf('Última fila de la matriz B:\n');
disp(B(end, :));
fprintf('Segunda columna de la matriz B:\n');
disp(B(:, 2));
fprintf('Última columna de la matriz B:\n');
disp(B(:, end));

Primera fila de la matriz B:
     1     2     3
Última fila de la matriz B:
     3     4     5
Segunda columna de la matriz B:
     2
     4
Última columna de la matriz B:
     3
     5
```

Ejercicio 54_01. Operaciones con matrices de numpy en Python

```
# Operaciones con matrices
C = np.array([[1, 2, 3], [1, 3, 4]])
D = np.array([[3, 3, 3], [4, 1, 2]])
print("Matriz C: ")
print(C)
print("Matriz D: ")
print(D)
print("Suma de C y D: %s" %(C+D))
print("Multiplicación de C y D: %s" %(C*D))

Matriz C:
     [[1  2  3] [1  3  4]]
Matriz D:
     [[3  3  3] [4  1  2]]
```

```
Suma de C y D:
     [[4   5   6] [5   4   6]]
Multiplicación de C y D: [[3   6   9] [4   3   8]]
```

Ejercicio 54_02. Operaciones con matrices en MATLAB

```
% Operaciones con matrices
C = [1, 2, 3; 1, 3, 4];
D = [3, 3, 3; 4, 1, 2];
fprintf('Matriz C:\n');
disp(C);
fprintf('Matriz D:\n');
disp(D);
fprintf('Suma de C y D:\n');
disp(C + D);
fprintf('Multiplicación de C y D:\n');
disp(C .* D);
```

```
Matriz C:
     1     2     3
     1     3     4
Matriz D:
     3     3     3
     4     1     2
Suma de C y D:
     4     5     6
     5     4     6
Multiplicación de C y D:
     3     6     9
     4     3     8
```

1.5.4. Funciones de utilidad

- **Funciones útiles para crear *arrays* en Python:**

Array vacío: np.empty(dimensiones)
Array relleno de ceros: np.zeros(dimensiones)
Array aleatorio: random.randint(tope, size=(dimensiones))

Documentación para: Vectores aleatorios

- **Funciones útiles para crear arrays en Matlab:**

Array vacío: v = []; % Crea un vector vacío usando corchetes vacíos
Array relleno de ceros: A = zeros(m, n); % Crea una matriz m x n de ceros
Array relleno de unos: A = ones(m, n); % Crea una matriz m x n de unos
Array aleatorio: A = randn(m, n); % Crea una matriz m x n de valores aleatorios normalmente distribuidos

Ejercicio 55_01. *Array* vacío especificando las dimensiones

```
# array vacío especificando las dimensiones 2x2:
import numpy as np
a = np.empty([2, 2])
```

```
print("Mostramos el resultado: ", a)
print("Mostramos dimensiones: ", a.shape)
# le damos valores
a[0, 0] = 1
a[0, 1] = 2
a[1, 0] = 3
a[1, 1] = 4
print("Mostramos tras dar valores: ", a)

Mostramos el resultado: [[0   0] [0   0]]
Mostramos dimensiones: (2, 2)
Mostramos tras dar valores: [[1.  2.] [3.  4.]]
```

Ejercicio 55_02. *Array* vacío especificando las dimensiones

```
% Array vacío especificando las dimensiones
a = zeros(2); % o a = []; para un array vacío
fprintf('Mostramos el resultado:\n');
disp(a);
fprintf('Mostramos dimensiones: %s\n', mat2str(size(a)));
% Le damos valores
a(1, 1) = 1;
a(1, 2) = 2;
a(2, 1) = 3;
a(2, 2) = 4;
fprintf('Mostramos tras dar valores:\n');
disp(a);

Mostramos el resultado
     0     0
     0     0
Mostramos dimensiones
     2     2
Mostramos tras dar valores:
     1     2
     3     4
```

Ejercicio 56_01. *Array* lleno de ceros

```
import numpy as np
a = np.zeros(2)
print("Vector resultado: ", a)
b = np.zeros([3, 4])
print("Matriz resultado: \n", b) #\n para salto de linea

Vector resultado: [0. 0.]
Matriz resultado:
    [[0.  0.  0.  0.]
    [0.  0.  0.  0.]
    [0.  0.  0.  0.]]
```

Ejercicio 56_02. *Array* lleno de ceros

```
% Array lleno de ceros
```

```
b = zeros(1, 2); % Vector
fprintf('Vector resultado:\n');
disp(b);
c = zeros(3, 4); % Matriz
fprintf('Matriz resultado:\n');
disp(c);

Vector resultado:
     0     0
Matriz resultado:
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Ejercicio 57_01. *Array* aleatorio Python

```
from numpy import random
# Vector de enteros aleatorio de 10 posiciones con números entre 1 y 50
v = random.randint(50, size=(10))
print("Vector: \n", v)
# Matriz de enteros aleatoria de 3x4 con números entre 1 y 10
M = random.randint(10, size=(3,4))
print("Matriz: \n", M)

Vector:
    29   4   3  27  39  47  7  29  24   1
Matriz:
     [[4   4   7   7]
      [2   6   3   8]
      [8   2   7   5]]
```

Ejercicio 57_02. *Array* aleatorio MATLAB

```
% Vector de enteros aleatorio de 10 posiciones con números entre 1 y 50
v = randi([1, 50], 1, 10);
fprintf('Vector:\n');
disp(v);
% Matriz de enteros aleatoria de 3x4 con números entre 1 y 10
M = randi([1, 10], 3, 4);
fprintf('Matriz:\n');
disp(M);

Vector:
    29     4     3    27    39    47     7    29    24     1
Matriz:
     4     4     7     7
     2     6     3     8
     8     2     7     5
```