



**CICLO 1**

[FORMACIÓN POR CICLOS]

# Fundamentos de **PROGRAMACIÓN**



Ingeni@  
Soluciones TIC



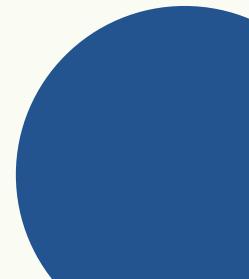
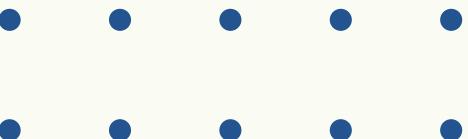
UNIVERSIDAD  
DE ANTIOQUIA

Facultad de Ingeniería

Lectura

# OPERACIONES

*con matrices: suma de filas y  
columnas, suma de matrices  
y recorridos especiales*



En la lectura anterior presentamos los algoritmos para recorrer e imprimir una matriz por filas. En general, las operaciones que se necesitan ejecutar con matrices requieren que la matriz se recorra por filas o por columnas. Una de estas operaciones es totalizar los datos de cada fila y de cada columna. En esta lectura presentamos los subprogramas con los cuales se efectúan dichas tareas.

## Suma de los datos de cada fila de una matriz.

Sumar los datos de cada fila es una operación que se requiere con frecuencia. Si consideramos la matriz del ejemplo del censo en varios departamentos, cada fila contiene el número de habitantes de cada municipio de cada departamento. Interesa conocer el total de habitantes de cada departamento: la suma de los datos de las celdas de la fila 1 da el total de habitantes del departamento 1; la suma de los datos de las celdas de la fila 2 da el total de habitantes del departamento 2; la suma de los datos de las celdas de la fila 3 da el total de habitantes del departamento 3, y así sucesivamente.

Para efectuar esta operación se necesita recorrer la matriz por filas, es decir, utilizaremos las instrucciones de recorrido de la matriz por filas, que fueron explicadas en el algoritmo **imprimeMatriz()**.

Una función que suma e imprime el total de los datos de las celdas de cada fila de una matriz es:

```
def sumarFilas(self):
    for i in range(1, self.m + 1):
        s = 0
        for j in range(1, self.n + 1):
            s = s + self.mat[i][j]
        print("El total de la fila ", i, " es: ", s)
```

Simplemente se usa un acumulador (**s**), que se inicializa en 0 cada vez que se va a empezar a recorrer una fila. Con el ciclo interno se recorre la fila **i**, y al terminar el ciclo interno se procede a mostrar el valor de **s**.



Sin embargo, es preferible que nuestro programa construya un vector en el que cada posición **i** del vector contenga la suma de los datos de la fila **i**. Un programa con esta característica es:

```
def sumarFilas(self):
    v = vector(self.m)
    for i in range(1, self.m + 1):
        s = 0
        for j in range(1, self.n + 1):
            s = s + self.mat[i][j]
        v.agregarDatos(s)
    return v
```

En esta función definimos un objeto de la clase **vector (v)** con un tamaño igual al número de filas que tenga la matriz. Tenga presente siempre que cada objeto de la clase matriz tiene sus propios valores de **m** y **n**: número de filas y número de columnas. Esa es una de las grandes ventajas de la POO. Cada objeto es dueño de sus propios atributos, siempre viajan con el objeto, no se requieren parámetros adicionales.

Como ejemplo, consideremos la matriz de la figura 1:

	1	2	3	4	5	6	7
1	3	1	6	2	8	4	5
2	9	7	7	4	1	2	4
3	5	6	2	8	5	4	7
4	3	5	5	4	8	9	3
5	8	4	5	8	8	7	6
6	6	3	5	9	8	4	2
7	2	5	7	5	4	4	1

**Figura 1**

Al ejecutar nuestro segundo programa, retorna un vector con los siguientes datos:

	0	1	2	3	4	5	6	7
V	7	29	34	37	37	46	37	28

## Suma de los datos de cada columna de una matriz.

Así como en muchas situaciones es necesario sumar los datos de las celdas de cada fila, hay otras en las cuales lo que se requiere es sumar los datos de las celdas de cada columna.

Para totalizar los datos de las celdas de cada columna de una matriz, basta con recorrer la matriz por columnas. Un algoritmo que ejecuta dicha tarea se presenta a continuación:

```
def sumarColumnas(self):
    v = vector(self.n)
    for i in range(1, self.n + 1):
        s = 0
        for j in range(1, self.m + 1):
            s = s + self.mat[j][i]
        v.agregarDatos(s)
    return v
```

Observe y analice bien la diferencia entre recorrer por filas y recorrer por columnas. En el recorrido por filas, el primer ciclo **for** la variable controladora del ciclo va desde 1 hasta **m + 1**, siendo **m** el número de filas de la matriz, mientras que en el recorrido por columnas el primer ciclo **for** va desde 1 hasta **n + 1**, siendo **n** el número de columnas de la matriz.

**Atención:** la instrucción para acumular los datos en el recorrido por filas es **s = s + mat[i][j]**, mientras que en el recorrido por columnas es **s = s + mat[j][i]**. En el recorrido por filas, para cualquier valor de la fila (**i**) se recorren todos los valores de la **columna(j)**, mientras que en el recorrido por columnas, para cualquier valor de la columna (**i**) se recorren todos los valores de la fila (**j**).



Al ejecutar esta función de sumar los datos de las columnas sobre la matriz de la figura 1 se obtiene el siguiente vector:

	0	1	2	3	4	5	6	7
V	7	36	31	37	40	42	34	28

## Suma de dos matrices.

Otra operación importante y frecuente con matrices es la suma de ellas. La suma de matrices simplemente es sumar los elementos correspondientes de cada posición, es decir, el dato de la fila 1 columna 1 de una matriz con el dato de la fila 1 columna 1 de la otra matriz; el dato de la fila 1 columna 2 de una matriz con el dato de la fila 1 columna 2 de la otra matriz. En general, el dato de la fila  $i$  columna  $j$  de una matriz con el dato de la fila  $i$  columna  $j$  de la otra matriz.

Para sumar dos matrices la única condición que se debe cumplir es que tengan el mismo número de filas y columnas.

Una función para efectuar dicha tarea es:

```
def __add__(self, b):
    c = matriz(self.m, self.n)
    if self.m != b.m or self.n != b.n:
        print("Matrices no se pueden sumar. Son de
dimensions diferentes")
        return c
    for i in range(1, self.m + 1):
        for j in range(1, self.n + 1):
            c.mat[i][j] = self.mat[i][j] + b.mat[i][j]
    return c
```

En nuestra función recargamos nuevamente el operador `+`. Se crea una nueva matriz (**c**), que tiene las mismas dimensiones de las matrices a sumar. Luego se controla que las matrices tengan las mismas dimensiones.

```
s the element inside the
a = w.scrollLeft();
b = w.scrollTop();
o = t.offset();
x = o.left;
y = o.top;

ax = settings.accX;
ay = settings.accY;
th = t.height();
wh = w.height();
tw = t.width();
ww = w.width();

(y + th + ay >= b &&
 y <= b + wh + ay &&
 x + tw + ax >= a &&
 x <= a + ww + ax) {

    //trigger the cust
    if (!t.appeared) t

} else {

    //it scrolled out
    t.appeared = false
}

//create a modified fn wi
or modifiedFn = function(
    //mark the element as
    t.appeared = true;
    ...
    //is this supposed to
    ...
    //is this supposed to
    ...
}
```

Recuerde que **self** se refiere al objeto que invocó el método. Entonces se controla que el numero de filas (**m**) y el número de columnas sean iguales, de lo contrario se produce el mensaje de que no se pueden sumar y retorna la matriz resultado vacía. Para sumar, simplemente se recorren simultáneamente, por filas, ambas matrices, sumando los datos uno a uno. Por último, retorna la matriz construida (**c**).

## Producto de los datos en la diagonal principal.

Una operación interesante en los procesos con matrices es el producto de los datos pertenecientes a la diagonal principal. Una función para este propósito es:

```
def productoDiagonalPrincipal(self):  
    f = 1  
    for i in range(1, self.m + 1):  
        f = f * self.mat[i][i]  
    return f
```

Es importante notar que para poder efectuar esta función la matriz debe ser cuadrada.