Universidad Tecnológica Nacional
Facultad Regional Córdoba

Algoritmos y Estructuras de Datos

Ficha Nro. 16

Arreglos Bidimensionales

Cursos 1k07-1k08 Docente: Ing. Corso Cynthia L.

Temario

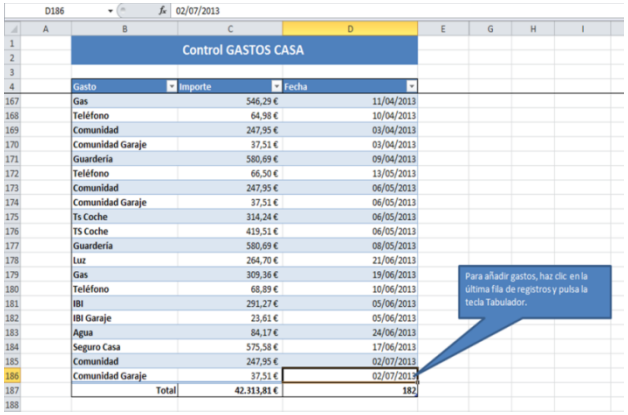
- Introducción
- Matrices (arreglos bidimensionales)
  - Creación
  - Carga de elementos de la matriz
  - Acceso de elementos
  - Recorridos. Por filas y por columnas
  - Sumatoria de elementos de la matriz por columna.

Introducción

- Arreglo unidimensional: es una estructura de datos que permite procesar un conjunto significativo de datos.

- Los problemas analizados hasta ahora fueron de naturaleza unidimensional (cantidad de índices para acceder a los elementos) y estos se utilizaron como medio de soporte para almacenar los datos.

- Existen otras situaciones en los que se necesita organizar los datos de manera que el acceso a los datos es necesario mediante dos o más índices.

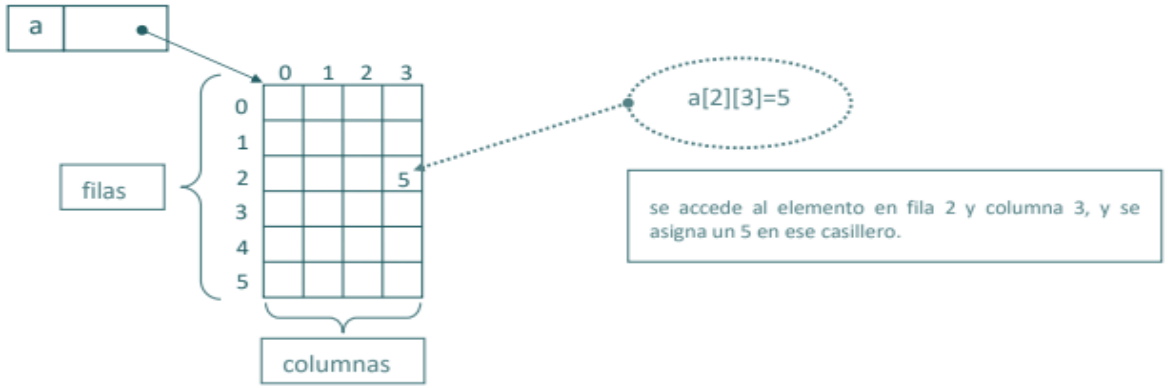


Matrices

- Concepto: es un arreglo bidimensional cuyos elementos están almacenados en forma de tabla conformado por filas (disposiciones horizontales) y columnas (disposiciones verticales).

- Para el acceso de cada elemento es necesario el uso de dos índices.

Figura 1: Esquema de representación de una matriz de orden 6*4.




```
a = [ [2, 3, 4, 5],
      [1, 5, 6, 3],
      [7, 2, 4, 0],
      [8, 4, 1, 7],
      [0, 0, 2, 9],
      [1, 3, 4, 1],
    ]

a[2][3] = 5  # accede a una casilla y cambia su valor.
```

Matrices: forma de implementación en Python

- En Python las matrices se implementan con el tipo list, usando el concepto de listas embebidas, ya que una list puede contener otras list embebidas en ella.

- Es decir que se implementan listas de listas (variable de tipo list, que en cada celda tiene a otra list).

Matrices: Creación (i)

Creación de la matriz con asignación directa de valores constantes. Existen dos posibilidades:

- Posibilidad 1: identación de cada fila:

```
m0 = [  [1, 3, 4],
        [3, 5, 2],
        [4, 7, 1]
      ]
print('Matriz con valores fijos:', m0)
```

- Posibilidad 2: sin indentación

```
m0 = [ [1, 3, 4], [3, 5, 2], [4, 7, 1] ]
print('Matriz con valores fijos:', m0)
```

Matrices: Creación (ii)

Creación de la matriz sin valores fijos, con "n" filas y "m" columnas (Donde "n" y "m" son variables).

En Python las matrices son construidas en tiempo de ejecución. Estudiaremos tres alternativas posibles:

- Alternativa 1: la mas descriptiva e intuitiva, pero la mas ineficiente.
- Alternativa 2: menos detallada e intuitiva, pero mas compacta que la anterior.
- Alternativa 3: mas compacta y eficiente, aunque posiblemente menos clara.

Matrices: Creación (iii)

• Alternativa 1: Supongamos se desea crear una matriz denominada m1 con "n" filas **n = 3** y "m" columnas **m = 2**.

1) La idea es partir de una lista vacía, y asignar en ella "n" listas vacías.

```
m1 = []
for f in range(n):
    m1.append([])
```

m1=> [[],[],[]]

2) Agregar en cada una de las "n" listas de m1, un total "m" de elementos.

```
m1 = []
for f in range(n):
    m1.append([])
    for c in range(m):
        m1[f].append(None)
```

Apariencia final **m1 = [** [None,None], [None,None], [None,None]**]**

Matrices: Creación (iv)

- Alternativa 2: Supongamos se desea crear una matriz denominada m2 con "n" **filas n = 3** y "m" **columnas m = 2.**

1) Se comienza creando "n" filas con el operador multiplicación.

```
m2 = [None] * n     => m2 = [ None, None, None ]
```

2) Para completar la matriz se itera sobre los elementos, , en este caso None, y se los reemplaza por una lista de "m" elementos.

```
m2= [None] * n
for f in range(n):
  # crea la columnas para cada fila
   m2[f] = [None] * m #Expande cada fila a 2 elem.
```

Apariencia final: m2 = **[** [None, None], [None, None], [None,None] **]**

Matrices: Creación (v)

- Alternativa 3: Supongamos se desea crear una matriz denominada m3 con "n" **filas n = 3** y "m" **columnas m = 2**. Para su creación se utiliza una expresión de comprensión.

```
m3 = [[None] * m for f in range(n)]
```

- El ciclo for realiza una iteración de "n" vueltas.
- En cada vuelta crea una lista de "m" elementos "None".
- Si el valor de filas es n = 2 y columnas es m = 3 la estructura de la matriz generada sería:

Apariencia final: m3 = [[None, None], [None, None], [None,None]]

Matrices: Acceso Individual

- El acceso individual en una matriz es mediante dos índices: el primero selecciona "la fila" (o sea una de las "n" sublistas) y el segundo selecciona "la columna" (o sea el elemento dentro de la sublista seleccionada con el primer índice). Ejemplos:

```
matriz[0][3] = 10
matriz[1][2] = 20
```

- El primer índice de cada dimensión es cero.
- Los índices de acceso a filas y columnas no puede sobrepasar al espacio reservado.
- Si se dispone una matriz de dimensión 2 x 2, la siguiente operación sería incorrecta:

```
a = matriz[0][4] #Error
```

Índice que sobrepasa la dimensión reservada para las columnas...

Matrices: Carga de elementos

- Se usan dos ciclos “for” uno para recorrer las filas y el otro para las columnas. De esta forma se accede a una celda en particular.

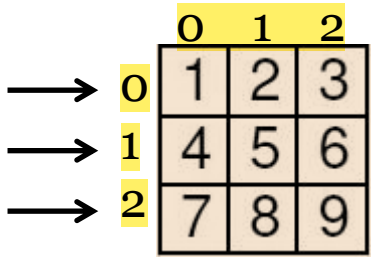
```
fil, col = 3, 3
mat = [[None] * col for f in range(fil)]
for f in range(len(mat)):
  for c in range(len(mat[0])):
    mat[f][c] = int(input('Nro:'))
```

len(mat) → Retorna cantidad de “filas”

len(mat[0]) → Retorna cantidad de “columnas”

Solicita al usuario por teclado un valor numérico y lo asigna en una casilla de una determinada “fila” y “columna”.

Ejemplo de matriz ya cargada.



En Python

```
mat = [[1,2,3],[4,5,6],[7,8,9]]
```

Matrices: Recorrido completo (i)

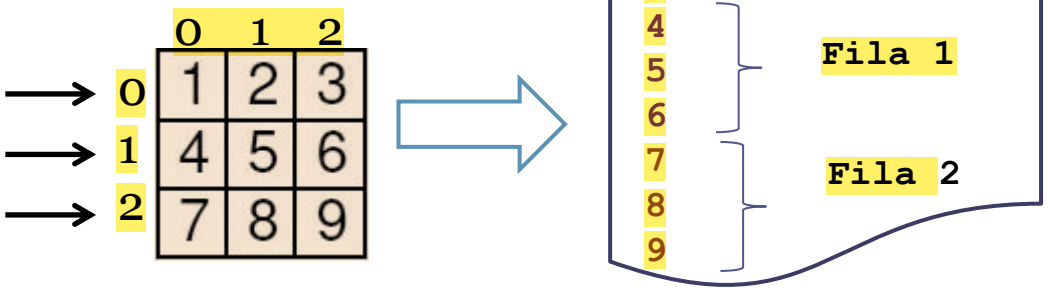
- Recorrido completo de la matriz por "filas": se usan dos ciclos "for" uno para recorrer las filas y el otro para las columnas.

```
# Se supone ya creada una matriz
# denominada mat de orden n * m.
for f in range(len(mat)):
    for c in range(len(mat[0])):
        print(mat[f][c])
```

len(mat) → Retorna cantidad de "filas"

len(mat[0]) → Retorna cantidad de "columnas"

Visualiza los elementos de la matriz por "filas"



Matrices: Recorrido completo (ii)

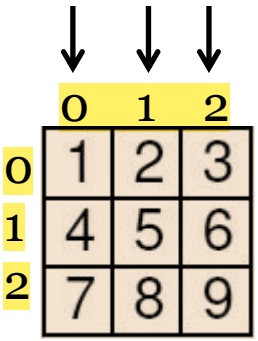
- Recorrido completo de la matriz por "columnas": se deben invertir los dos ciclos "for", llevar mas afuera el que recorre las columnas y mas interno el que recorre las filas.

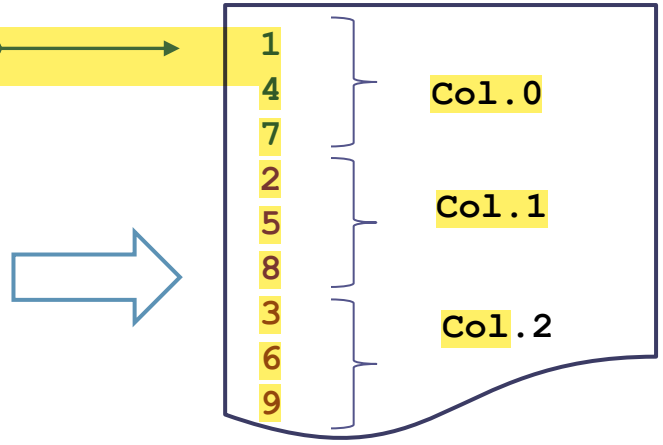
```
# Se supone ya creada una matriz
# denominada mat de orden n * m.
for c in range(len(mat[0])):
    for f in range(len(mat)):
        print(mat[f][c])
```

len(mat[0])→ Retorna cantidad de "columnas"

len(mat)→ Retorna cantidad de "filas"

Visualiza los elementos de la matriz por "columnas"





Matrices: Recorrido completo (iii)

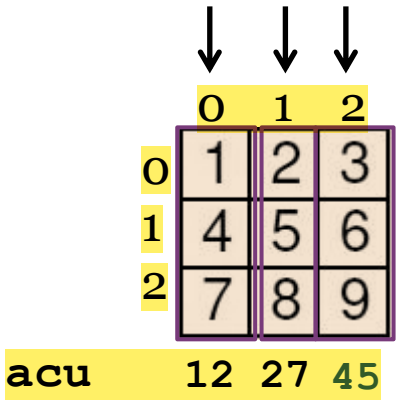
A tener en cuenta!!!

La variable que indica la fila va en el primer par de corchetes, y la variable que indica la columna va en el segundo par de corchetes, sin importar cuál ciclo va por fuera y cuál por dentro.

Matrices: Totalización de elementos por columnas

```
# Script de carga por teclado de la matriz "mat"
filas, columnas = 3, 3
mat = [[0] * columnas for f in range(filas)]
for c in range(columnas):
  for f in range(filas):
      mat[f][c] = int(input('Valor: '))
print('Matriz leída:', mat)
```

```
# Script para totalizar los elementos de la matriz "mat"
acu = 0
for c in range(columnas):
  for f in range(filas):
      acu+ = mat[f][c]
print('Acumulación de datos de mat:', acu)
```



Bibliografía

- [1] V. Frittelli, Algoritmos y Estructuras de Datos, Córdoba: Universitas, 2001.
- [2] Python Software Foundation, "Python Documentation," 2015. [Online]. Available: https://docs.python.org/3/. [Accessed 24 February 2015].
- [3] M. Pilgrim, "Dive Into Python - Python from novice to pro," 2004. [Online]. Available: http://www.diveintopython.net/toc/index.html. [Accessed 6 March 2015].