Introducción a las matrices en Python

La matrices no son una estructura propia de Python. Simplemente, una matriz es una lista de listas que nosotros interpretamos desde el punto de vista matemático. Es decir, la estructura `m = [[1,2],[3,4]]` nosotros la interpretamos como la matriz 2x2 cuya primera fila es (1,2) y cuya segunda fila es (3,4), pero esto no deja de ser una interpretación.

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

Out[1]:
```
[[1, 2], [3, 4]]
```

Para representar una matriz, debemos crear una función específica.

In [2]:
```
def dibujaMatriz(M):
    for i in range(len(M)):
        print '[',
        for j in range(len(M[i])):
            print '{:>3s}'.format(str(M[i][j])),
        print ']'

dibujaMatriz(m)
```

```
[   1   2 ]
[   3   4 ]
```

In [3]:
```
n = [[1,10,100], [20,2,200],[300,30,3]]
dibujaMatriz(n)
```

```
[   1  10 100 ]
[  20   2 200 ]
[ 300  30   3 ]
```

Para representar una matriz, a veces es interesante pasarlo a una cadena de caracteres

In [4]:
```
def matriz2str(matriz):
    cadena = ''
    for i in range(len(matriz)):
        cadena += '['
        for j in range(len(matriz[i])):
            cadena += '{:>4s}'.format(str(matriz[i][j]))
        cadena += ']\n'
    return cadena
```

In [5]:
```
s = matriz2str(n)
print s
```

```
[   1  10 100]
[  20   2 200]
[ 300  30   3]
```

Podemos crear matrices de diversas maneras

In [6]:
```
def creaMatriz(n,m):
    '''
    Esta función crea una matríz vacía con n filas y n columnas.
    @param n : Número de filas.
    @param m : Número de columnas
    @type n: int
    @type m: int
    @return: devuelve una matriz n por m
    @rtype: matriz (lista de listas)
    '''
    matriz = []
    for i in range(n):
        a = [0]*m
        matriz.append(a)
    return matriz
```

In [7]:
```
def creaMatrizDato(n,m, dato):
    '''
    Esta función crea una matríz con n filas y n columnas.
    Cada celda contiene el valor "dato"
    @param n : Número de filas.
    @param m : Número de columnas
    @param dato: Un valor
    @type n: entero
    @type m: entero
    @type dato: tipo simple
    @return: devuelve una matriz n por m
    @rtype: matriz (lista de listas)
    '''
    matriz = []
    for i in range(n):
        a = [dato]*m
        matriz.append(a)
    return matriz
```

Cuidado: hay que crear bien las matrices

In [8]:
```
def badmatrix(n,m):
    a = [0]*m
    matriz = [a]*n
    return matriz

M = badmatrix(2,2)
print M
```

```
[[0, 0], [0, 0]]
```

In [9]:
```
M[0][0]=1
print M
```

```
[[1, 0], [1, 0]]
```

Dada una matriz, podemos estudiar si es correcta

In [10]:
```
def matrizCorrecta(M):
    '''
    Nos dice si una matriz es correcta.
    @param M: una matriz
    @type M: matriz
    @return: True si es correcta, False en caso contrario
    '''
    filas = len(M)
    columnas = len(M[0])
    correcto = True
    i = 1
    while i < filas and correcto:
        correcto = (len(M[i]) == columnas)
        i += 1
    return correcto
```

In [11]:
```
M = [[1,2,3], [2,4]]
matrizCorrecta(M)
```

Out[11]:
```
False
```

Puede ser útil la utilización de funciones auxiliares

In [12]:

```
def filas(M):
    '''
    Nos dice el número de filas de una matriz correcta.
    @param M: una matriz
    @type M: matriz
    @return: número de filas
    '''
    if matrizCorrecta(M):
        return len(M)

def columnas(M):
    '''
    Nos dice el número de columnas de una matriz correcta.
    @param M: una matriz
    @type M: matriz
    @return: número de columnas
    '''
    if matrizCorrecta(M):
        return len(M[0])

def matrizIdentidad(n):
    '''
    Crea una matriz identidad de tamaÃ±o n
    @param n : número de filas.
    @type n : entero
    @return: matriz identidad de tamaÃ±o n
    '''
    m = creaMatriz(n,n)
    for i in range(n):
        m[i][i] = 1
    return m

def copy(m):
    '''
    Realiza una copia independiente de la matriz
    '''
    result=[]
    for f in m:
        result.append(f[:])
    return result
```

Recuerda que no se puede leer una matriz directamente del teclado, utilizando `raw_input()` . Se puede leer una matriz introduciendo por teclado cada una de sus entradas

In [13]:
```
def leeMatriz(n,m):
    '''
    Esta función lee por teclado una matríz con n filas y n columna
s.
    @param n : Número de filas.
    @param m : Número de columnas
    @type n: entero
    @type m: entero
    @return: devuelve una matriz n por m
    '''
    A = creaMatriz(n,m)
    for i in range(n):
        for j in range(m):
            A[i][j] = int(raw_input('Introduce la componente (%d,%d
): '%(i,j)))
    return A
```

Puede ser más comodo, para matrices grandes, si se lee desde un fichero **(ejercicio)**

In [14]:
```
def copy(m):
    '''
    Crea una copia de la matriz
    '''
    result=[]
    for f in m:
        result.append(f[:])
    return result
```

Algunas operaciones matemáticas con matrices

In [15]:
```
def sumaMatriz(A,B):
    '''
    Suma dos matrices. Las dos matrices deben ser de la misma dimen
siÃ³n
    @param A: una matriz nxm
    @param B: una matriz nxm
    @type A: Matriz
    @type B: Matriz
    @return: Matriz suma
    '''
    if filas(A) == filas(B) and columnas(A) == columnas(B):
        C = creaMatriz(filas(A), columnas(A))
        for i in range(filas(A)):
            for j in range(columnas(A)):
                C[i][j] = A[i][j] + B[i][j]
        return C
```

In [16]:
```
def multiplicaMatriz(A,B):
    '''
    Multiplica dos matrices. El número de columnas de la primera de
be ser igual al número de filas de la segunda.
    @param A: una matriz nxm
    @param B: una matriz mxk
    @type A: Matriz
    @type B: Matriz
    @return: Matriz multiplicación nxk
    '''
    if columnas(A) == filas(B):
        C = creaMatriz(filas(A), columnas(B))
        for i in range(filas(C)):
            for j in range(columnas(C)):
                for k in range(columnas(A)):
                    C[i][j] += A[i][k] * B[k][j]
        return C
```

In [17]:
```
def traspuesta(M):
    '''
    Calcula la matriz traspuesta de M
    '''
    m = len(M) #filas
    n = len(M[0]) # columnas
    T = creaMatriz(n,m)
    for i in range(n):
        for j in range(m):
            T[i][j] = M[j][i]
    return T
```

Determinantes

Podemos calcular el determinante de una matriz cuadrada de dos maneras. La primera es tratar de transformar la matriz en otra en la que solo hay ceros debajo de la diagonal principal. La matriz transformada tiene el mismo determinante que la matriz original (salvo tal vez el signo).

Para ejecutar las operaciones básicas, creamos unas funciones auxiliares.

In [18]:
```
def multiplicaFila(m,f,e):
    '''
    Multiplica la fila f por el valor e
    '''
    n=len(m)
    for c in range(n):
        m[f][c]=m[f][c]*e
```

In [19]:
```
m=[[2,1,3],[4,2,3],[2,3,2]]
```

In [20]:
```
multiplicaFila(m, 1, 3)
m
```

Out[20]:
```
[[2, 1, 3], [12, 6, 9], [2, 3, 2]]
```

In [21]:
```
def combinacion(m,i,j,e):
    '''
    Combina las filas i y j, añadiendo a la fila j el producto de l
a
    fila i por un factor e
    '''
    n=len(m)
    for c in range(n):
        m[j][c]=m[j][c]+e*m[i][c]
```

In [22]:
```
combinacion(m, 0, 1, 10)
m
```

Out[22]:
```
[[2, 1, 3], [32, 16, 39], [2, 3, 2]]
```

In [23]:
```
def intercambiaFilas(m,i,j):
    m[i],m[j] = m[j],m[i]
```

In [24]:
```
intercambiaFilas(m, 0, 1)
m
```

Out[24]:
```
[[32, 16, 39], [2, 1, 3], [2, 3, 2]]
```

In [25]:
```
def determinante(matriz):
    '''
    Calcula el determinante poniendo ceros debajo
    de la diagonal principal
    '''
    m = copy(matriz)
    n=len(m)
    det=1
    for i in range(n):
        j=primeroNoNulo(m,i)
        if j == n:
            return 0
        if i!=j:
            det=-1*det
            intercambiaFilas(m,i,j)
        det=det*m[i][i]
        multiplicaFila(m,i,1./m[i][i])
        for k in range(i+1,n):
            combinacion(m,i,k,-m[k][i])
    return det

def primeroNoNulo(m,i):
    '''
    A partir de la fila i, busca la primera fila j cuya entrada
    (i,j) es nula
    '''
    result=i
    while result<len(m) and m[result][i]==0:
        result=result+1
    return result
```

In [26]:
```
mat = [[0, 1, 3], [1, 2, 3], [2, 0, 1]]
determinante(mat)
```

Out[26]:
```
-7.0
```

In [27]:
```
a = matrizIdentidad(4)
a
```

Out[27]:
```
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]
```

In [28]:
```
determinante(a)
```

Out[28]:
```
1.0
```

In [29]:
```
b = creaMatrizDato(5,5,1)
```

In [30]:
```
determinante(b)
```

Out[30]: 0

Otra posible solución, sería calcular *menores* de la matriz original y calcular el determinante a partir del cálculo de determinantes de matrices más pequeñas.

In [31]:
```
def menor(A,f,c):
    '''
    Calcula el "menor" que se obtiene a partir de A al quitar la fi
la f y la
    columna c.
    Suponemos que A es cuadrada
    '''
    if filas(A) == columnas(A):

        m = filas(A)
        M = creaMatriz(m-1, m-1)
        '''
        Dividimos la matriz en cuatro trozos
        [1 | 2]
        [3 | 4]
        '''
        # 1
        for i in range(f):
            for j in range(c):
                M[i][j] = A[i][j]

        # 2
        for i in range(f):
            for j in range(c,m-1):
                M[i][j] = A[i][j+1]
        # 3
        for i in range(f,m-1):
            for j in range(c):
                M[i][j] = A[i+1][j]
        # 4
        for i in range(f,m-1):
            for j in range(c,m-1):
                M[i][j] = A[i+1][j+1]

    return M
```

In [32]:
```
def determinante_rec(matriz):
    '''
    Calcula el determinante de forma recursiva, calculando los
    sucesivos menores
    '''
    if len(matriz) == 1:
        result = matriz[0][0]
    elif len(matriz) == 2:
        result = matriz[0][0]*matriz[1][1] - matriz[0][1]*matriz[1]
[0]
    else:
        result = 0
        i = 0
        sig = +1
        while i < len(matriz):
            mm = menor(matriz, i, 0)
            result += sig * matriz[i][0]* determinante_rec(mm)
            sig = - sig
            i += 1
    return result
```

In [33]:
```
determinante_rec(mat)
```

Out[33]: -7

Recursión? ---> eso qué es?

In [33]: