Listas de listas

Clase #13

IIC1103 – Introducción a la Programación

Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

- Listas de listas
- Matrices
- Ejercicios

Las listas pueden estar compuestas de listas

- Una lista puede contener otras listas. Por ejemplo, puedo tener una lista con las listas de películas favoritas de mis amigos:

- Ejemplo:

```
amigos = ["Hugo", "Paco", "Luis"]

peliculas_favoritas = [
    ["Gladiador", "Reto al destino", "El secreto de tus ojos"],
    ["Sherk", "Sherk 2", "Sherk 3", "Sherk 4"],
    ["El Señor de los Anillos: la comunidad del anillo",
     "El Señor de los Anillos: las dos torres",
     "El Señor de los Anillos: el retorno del rey"] ]

for i in range(len(amigos)):
    a = amigos[i]
    pf = peliculas_favoritas[i]
    print("Las películas favoritas de mi amigo", a, "son:")
    for pelicula in pf:
        print("-", pelicula)
```

Matrices en Python

¿Cómo podemos definir en general matrices en Python?

Podemos hacer uso de listas de listas, con las siguientes consideraciones:

- Una matriz **A=[$a_{i,j}$]** de **m** filas y **n** columnas, se representa como una lista que contiene **m** listas de **n** elementos
- El elemento **i,j** de **A** estará en la posición **[j-1]** de la lista **A[i-1]**
- Se puede acceder a él usando un doble índice: **A[i-1][j-1]**

- Usualmente, se espera que los elementos de la matriz sean todos del mismo tipo.

- En el caso de querer definir/usar funciones para operar matrices, éstas tienen que estar bien definidas.
- Por ejemplo: la suma entre matrices tiene sentido cuando las matrices que se suman tienen iguales dimensiones.

Suma de matrices

- Para $A=[a_{i,j}]$ y $B=[b_{i,j}]$ matrices de igual dimensión, se define la matriz C suma de A y B, esto es, $C = A+B$ como:
  - $C = [c_{i,j}]$ donde $c_{i,j} = a_{i,j} + b_{i,j}$

- En Python podemos implementar esta operación a través de la siguiente función:

```
def SumaMatrices(A,B):
    m = int(len(A))
    n = int(len(A[0]))
    C = []
    for i in range(0,m):
        fila = []
        for j in range(0,n):
            fila.append(A[i][j]+B[i][j])
        C.append(fila)
    return C
```

Multiplicación de matrices

- Para $A=[a_{i,k}]$ y $B=[b_{k,j}]$ matrices donde el número de columnas de la matriz **A** coincide con el número de filas de la matriz **B**, se define la matriz **C** producto de **A** y **B**, esto es, $C = A*B$ como:
  - $C = [c_{i,j}]$ donde $c_{i,j} = \Sigma(a_{i,k} * b_{k,j})$

- En Python podemos implementar esta operación a través de la siguiente función:

```
def MultiplicaMatrices(A,B):
    m = int(len(A))
    n = int(len(A[0]))
    p = int(len(B[0]))
    C = []
    for i in range(0,m):
        fila = []
        for j in range(0,p):
            suma = 0
            for k in range(0,n):
                suma += A[i][k]*B[k][j]
            fila.append(suma)
        C.append(fila)
    return C
```

Ejercicio 1 – cuadrado latino

- Un cuadrado latino de tamaño **`n`** x **`n`** es una matriz cuadrada cuyos valores son números naturales entre **`1`** y **`n`** (inclusive), en la cual cada fila y cada columna contiene todos los números entre **`1`** y **`n`**. Por ejemplo, el siguiente es un cuadrado latino de tamaño 5 x 5:

1	4	2	5	3
5	3	1	4	2
4	2	5	3	1
3	1	4	2	5
2	5	3	1	4

- Se te pide programar la función **`es_cuadrado_latino`** que recibe una matriz y retorna **`True`** o **`False`** dependiendo si dicha matriz representa o no un cuadrado latino.

Ejercicio 1 – cuadrado latino

- En este caso, usaremos matrices (listas de listas): **`matriz[i][j]`**

1	4	2	5	3
5	3	1	4	2
4	2	5	3	1
3	1	4	2	5
2	5	3	1	4

- Tenemos que recorrer cada fila y ver si están todos los números.
  - Fila `i`: **`matriz[i][0], matriz[i][1], . . ., matriz[i][n-1]`**

- Tenemos que recorrer cada columna y ver si están todos los números.
  - Columna `j`: **`matriz[0][j], matriz[1][j], . . ., matriz[n-1][j]`**

Ejercicio 1 – cuadrado latino

```
def es_cuadrado_latino(matriz):
    n = len(matriz)
    esCuadradoLatino = True

    # verificamos en cada fila
    for i in range(0,n):
        fila = matriz[i]
        for num in range(1,n+1):
            if not num in fila:
                esCuadradoLatino = False

    # verificamos en cada columna
    for j in range(0,n):
        columna = []
        for i in range(0,n):
            columna.append(matriz[i][j])
        for num in range(1,n+1):
            if not num in columna:
                esCuadradoLatino = False

    return esCuadradoLatino
```

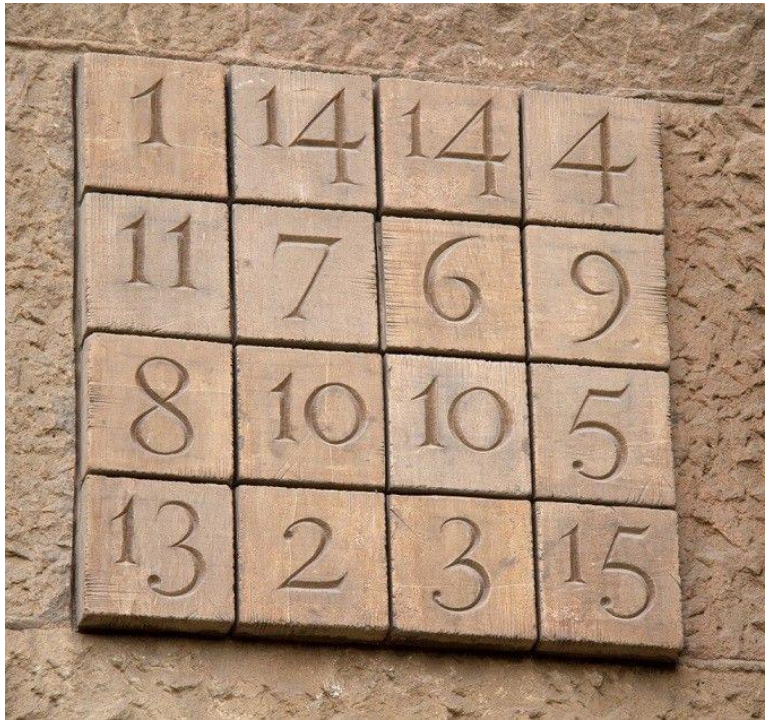
Ejercicio 1 – cuadrado latino

```
def MuestraMatriz(A):
    m = int(len(A))
    for i in range(0,m):
        print(A[i])

cuadrado = [ [1, 4, 2, 5, 3],
             [5, 3, 1, 4, 2],
             [4, 2, 5, 3, 1],
             [3, 1, 4, 2, 5],
             [2, 5, 3, 1, 4] ]

print("cuadrado:")
MuestraMatriz(cuadrado)
print("¿Es cuadrado latino?", es_cuadrado_latino(cuadrado))
```

```
>>>
cuadrado:
[1, 4, 2, 5, 3]
[5, 3, 1, 4, 2]
[4, 2, 5, 3, 1]
[3, 1, 4, 2, 5]
[2, 5, 3, 1, 4]
¿Es cuadrado latino? True
```

Ejercicio 2 – cuadrado mágico

- Un cuadrado mágico de tamaño **`n`** x **`n`** es una matriz cuadrada cuyos valores son números naturales, tal que al sumar los números en cada fila, en cada columna, y en cada diagonal, siempre se obtiene el mismo valor.
- Usualmente los números empleados para rellenar las casillas son consecutivos, de **`1`** a **n^2**.



Fuente: http://todosobrebarcelona.com/wp-content/uploads/2017/01/cuadro-del-33-de-gaudi-sagrada-familia.jpg

Material extra

TUPLAS

Tupla

- Una **tupla** es una secuencia de valores agrupados.
  - Sirve para agrupar datos, como si fueran un único valor, que por su naturaleza deben ir juntos.
- Una tupla puede ser creada poniendo los valores separados por comas y entre paréntesis.

- Observaciones
  - El tipo asociado a una tupla en Python se conoce como **`tuple`**.
  - El tipo **`tuple`** es inmutable, es decir una tupla no puede ser modificada una vez que ha sido creada.

- Ejemplos:
  - Los datos de una persona (nombre, apellido, rut)

```
persona = ("juan", "perez", "123456789-X")
```

  - Las coordenadas de un punto P en el espacio (x, y, z)

```
p = (5, 2, -3)
```

Tuplas – índices

- Al igual que las listas o String, los elementos de una tupla se identifican por índices.

- Ejemplos:

```
persona[0] → "juan"
persona[1] → "perez"
p[0] → 5
p[1] → 2
p[2] → -3
```

- Hemos usado tuplas, de manera implícita, para hacer las asignaciones de valores múltiples retornados por una función a un conjunto de variables.

- Link:
  - https://docs.python.org/3/library/stdtypes.html?highlight=tuple#tuples

Tuplas compuestas

- Una tupla puede contener otras tuplas. Por ejemplo, una persona puede ser descrita por su nombre, su rut y su fecha de nacimiento:

- Ejemplos:

```
candidato1 = ("Michelle Bachelet","5.811.892-3 " ,(29, 9, 1951))
candidato2 = ("Franco Parisi","6.872.197-0" ,(25,8,1967))
candidato3 = ("Evelyn Matthei","7.342.646-4",(11,11, 1953))
```

Listas con tuplas

- Una lista, como se ha dicho, puede contener elementos de cualquier tipo de datos.

- Un ejemplo de matemáticas
  - Supongamos que se tiene una lista L con una secuencia de puntos del espacio tridimensional, en formato de tuplas, esto es:

    `L=[(x0, y0, z0), (x1, y1, z1), … , (xn, yn, zn)]`

  - ¿Cómo se puede determinar que par de puntos (A,B) de L están más cercanos el uno del otro?

  - Como idea base de algoritmo debemos hacer un “doble” recorrido sobre la lista, esto es “todos con todos” para buscar que pareja de puntos tiene la distancia menor entre ellos.

Ejemplo – puntos más cercanos

```
import math

def Distancia(p, q):
    return math.sqrt((p[0]-q[0])**2+(p[1]-q[1])**2+(p[2]-q[2])**2)

def MasCercanos(L):
    p1=L[0]
    p2=L[1]
    distanciaMinima = Distancia(p1, p2)
    for punto1 in L:
        for punto2 in L:
            if punto1 != punto2:
                distancia = Distancia(punto1,punto2)
                if distancia < distanciaMinima:
                    p1 = punto1
                    p2 = punto2
                    distanciaMinima = distancia
    return [p1,p2] # podriamos tambien retornar la tupla (p1,p2)

lMC = MasCercanos([(2,0,0), (1,1,1), (1,1,2), (-2,0,0)])
print(lMC)
```

```
>>>
[(1, 1, 1), (1, 1, 2)]
```