Listas de Listas

Clase #20

IIC1103 – Introducción a la Programación

El plan de hoy es...

- Belen -> DCCSombrero
- Listas/Listas de listas
- Canvas-> Paginas -> Ejercicios resueltos (video)

- Bele
- Lista
- Canv

- La prueba es presencial, comienza a las 18:30 hrs. y dura 3 horas, es decir, termina a las 21:30 hrs. **IMPORTANTE:** No se dejará entrar a nadie que llegue más de 20 minutos tarde, así que procuren ser puntuales ⏰.
- Deben llevar una **identificación física** (TUC, Cédula de Identidad, Pase Escolar, etc.). No sirve mostrar el Portal UC o una identificación digital.
- Deben llevar sus computadores o tablets (con teclado) cargados. Habrán alargadores en las salas, aunque dada la gran cantidad de alumnos, no todos podrán estar conectados simultáneamente, por lo que deberán turnarse.
- La prueba sigue el mismo formato de los simulacros, esto es, deberán resolver una serie de ejercicios en Replit, descargar el código y subir cada parte en el buzón de Canvas correspondiente.
- Los buzones de Canvas para subir sus códigos **tendrán un único intento**, así que deben procurar subir el archivo correcto. **IMPORTANTE:** Solo se considerará el código que suban en Canvas, y se revisará que coincida con su trabajo en Replit 👀.
- Dado que hemos hecho varios simulacros, ejercicios y ayudantías en Replit, es su responsabilidad tener acceso al "team" del curso en esa plataforma y conocer su funcionamiento. Si aún no tienen acceso, deben escribir lo antes posible a iic1103@uc.cl. **IMPORTANTE:** Si el día de la prueba aún no se han integrado al curso en Replit, no se garantiza que se les añada rápidamente y, en ningún caso, se les devolverá el tiempo que hayan perdido ⚠️.
- En línea con lo anterior, también es su responsabilidad **contar con acceso al internet de la universidad** ("Eduroam").
- La prueba es individual y **solo pueden tener abierto las preguntas de la prueba en Replit** y el buzón de Canvas cuando vayan a subir sus preguntas.
- Se les entregará un resumen de algunos contenidos en formato físico, que NO deben rayar, pues deben devolverlo al final de la prueba.

Problema #1

- Queremos guardar la fecha de cumpleaños y mail de nuestro amigo

Iñigo Montoya	iñigo@gmail.com	03/10/1995

- Ejemplo:
- `lista = [“Iñigo Montoya”, “iñigo@gmail.com”,”03/10/1995”]`

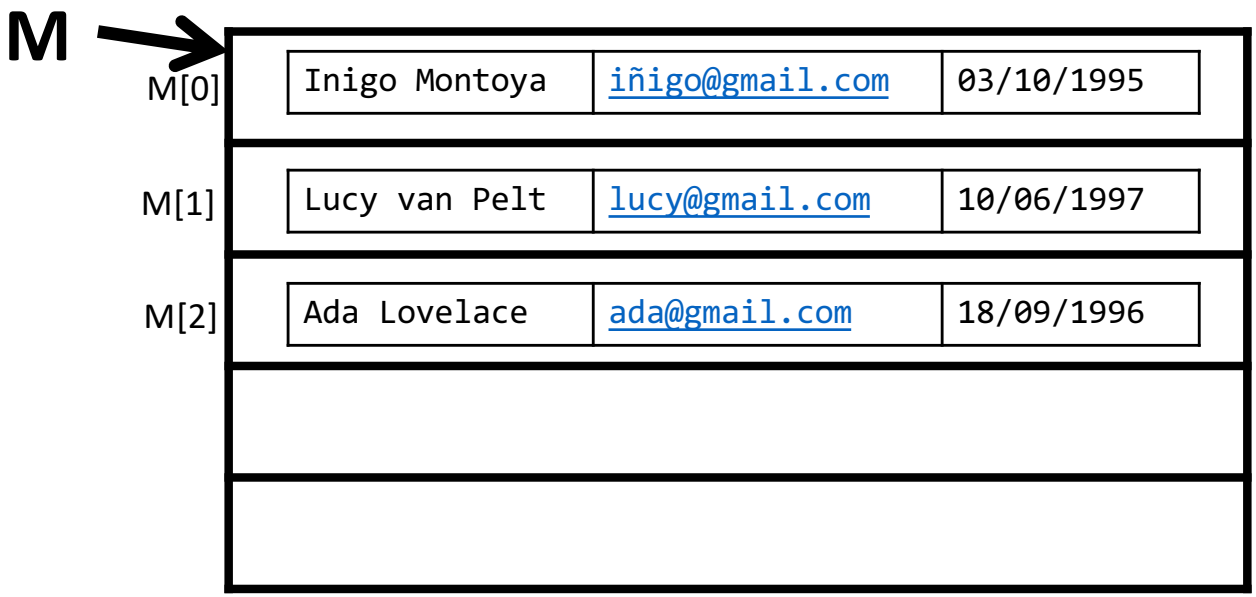
Problema #2

- Pero: no tengo un solo amigo!

Inigo Montoya	iñigo@gmail.com	03/10/1995
Lucy van Pelt	lucy@gmail.com	10/06/1997
Ada Lovelace	ada@gmail.com	18/09/1996
...	...	...

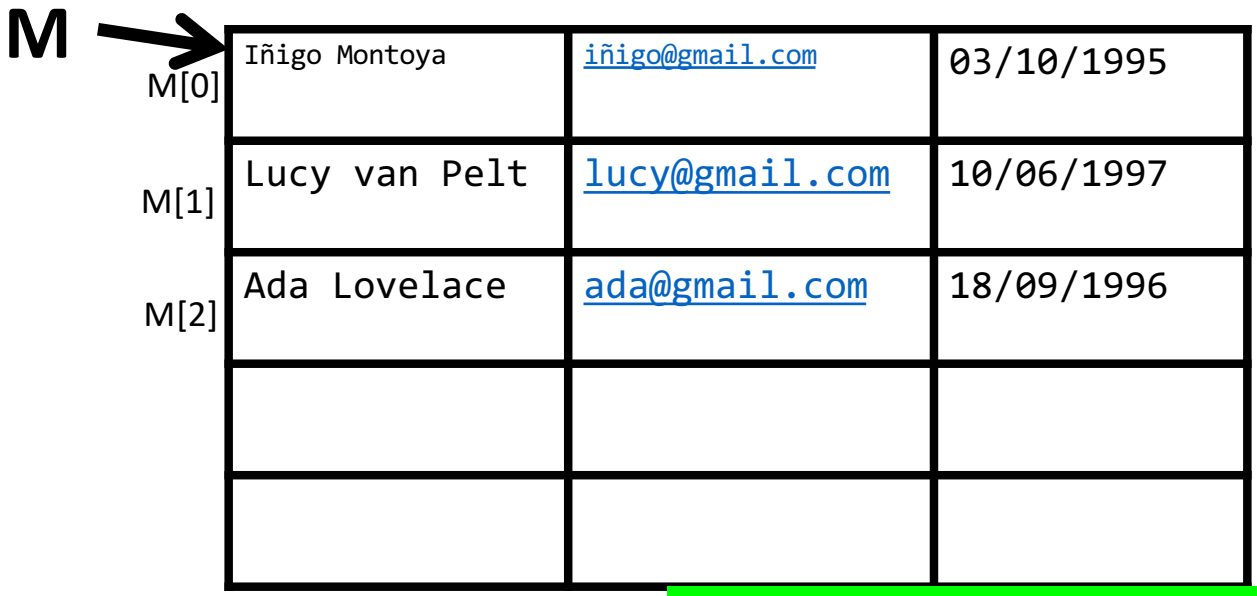
Problema #2

- Queremos guardar los datos de todos nuestros amigos
- … necesitamos una **lista de listas** (o matriz)



Problema #2

- Queremos guardar los datos de todos nuestros amigos
- … necesitamos una **lista de listas** (o matriz)



¿Tengo amigos de cumpleaños el 25 de diciembre?

¿Cuáles amigos tienen cumpleaños en Junio?

¿Cuál es el mail de Lucy?

¿Cuándo es el cumpleaños de Ada?

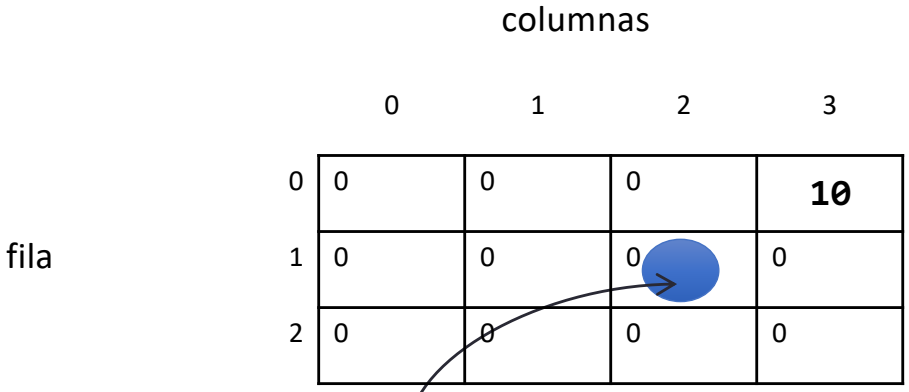
Listas de listas (Listas multidimensionales)

columnas

fila	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

- `M=[[0,0,0,0],`
- `  [0,0,0,0],`
- `  [0,0,0,0]]`

Listas de listas (Listas multidimensionales): Acceder a una celda



- `M[1][2] #fila – columna`
- `Ej:`
- `M[0][3] = 10`

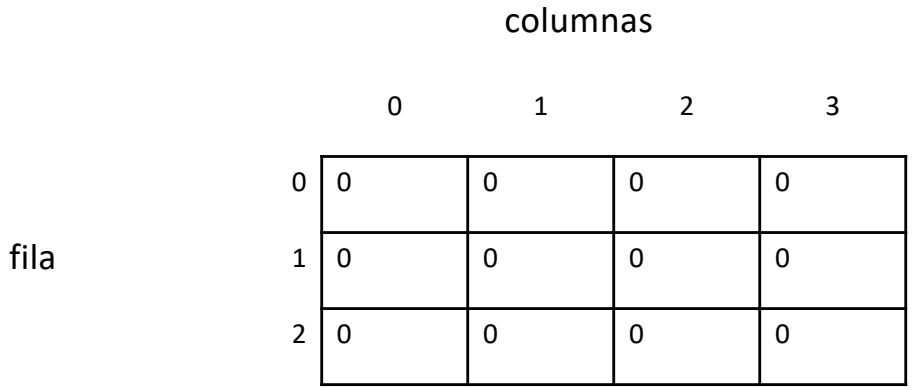
Listas de listas (Listas multidimensionales): Tamaño

columnas

fila	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

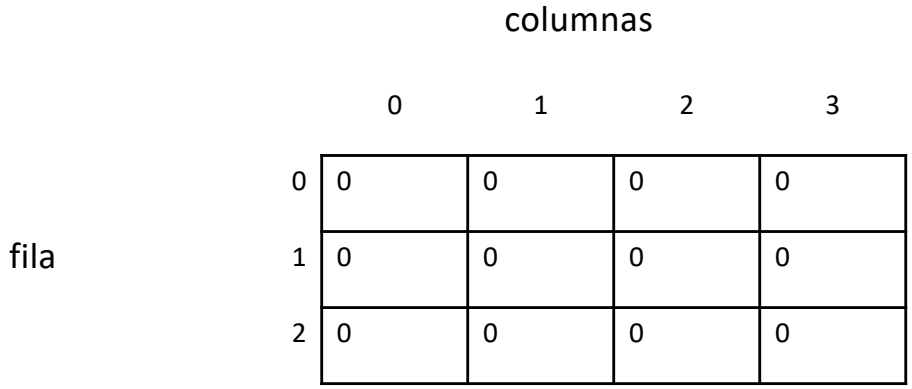
- ¿Número de filas de M?
- ¿Número de columnas de M?

Listas de listas (Listas multidimensionales) : Tamaño



- ¿Número de filas de M? **len(M)**
- ¿Número de columnas de M? **len(M[0])**

Listas de listas (Listas multidimensionales) : Creación



valor cols filas

- `M= [[0 for x in range(4)] for x in range(3)]`

Aprender más: List comprehensions

- https://www.w3schools.com/python/python_lists_comprehension.asp
- `lista = [expresion for item in iterable if condicion == True]`

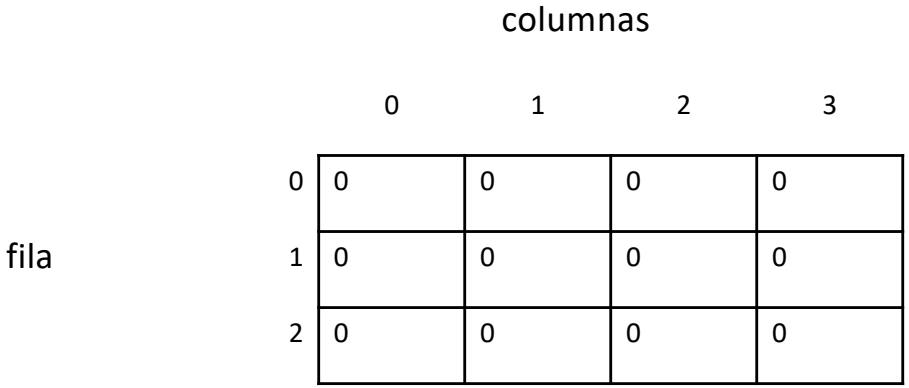
- `>>> a = ['lunes','martes','miercoles','jueves','viernes','sabado','domingo']`
- `>>> b = [x for x in a if x[0]=='m’]`
- `>>> c = ['x' for y in a]`

Listas de listas (Listas multidimensionales) : Recorrer

columnas

fila	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

Listas de listas (Listas multidimensionales) : Recorrer



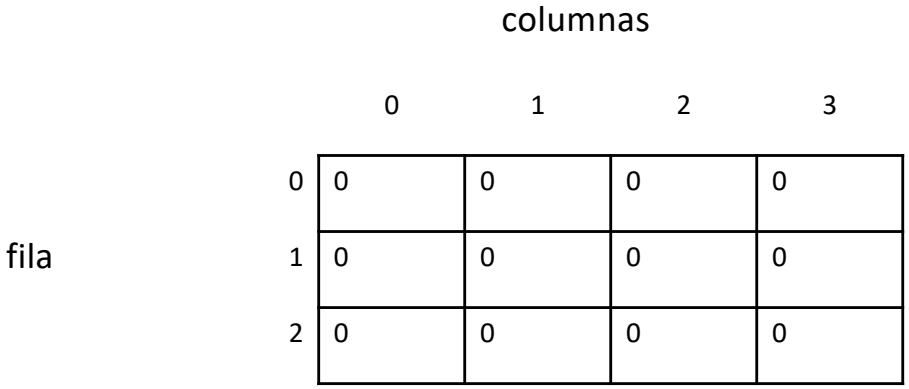
- M= ...
- ```
  for i in range(len(M)):
      for j in range(len(M[0])):
          print(M[i][j],end=" ")
      print()
  ```
  ```

Listas de listas (Listas multidimensionales) : Buscar

columnas

fila	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

Listas de listas (Listas multidimensionales) : Buscar



- M= ...
-
```
x = int(input("Qué quieres buscar?"))
for i in range(len(M)):
    for j in range(len(M[0])):
        if M[i][j]==x:
            print("Lo encontré en",i,j)
```

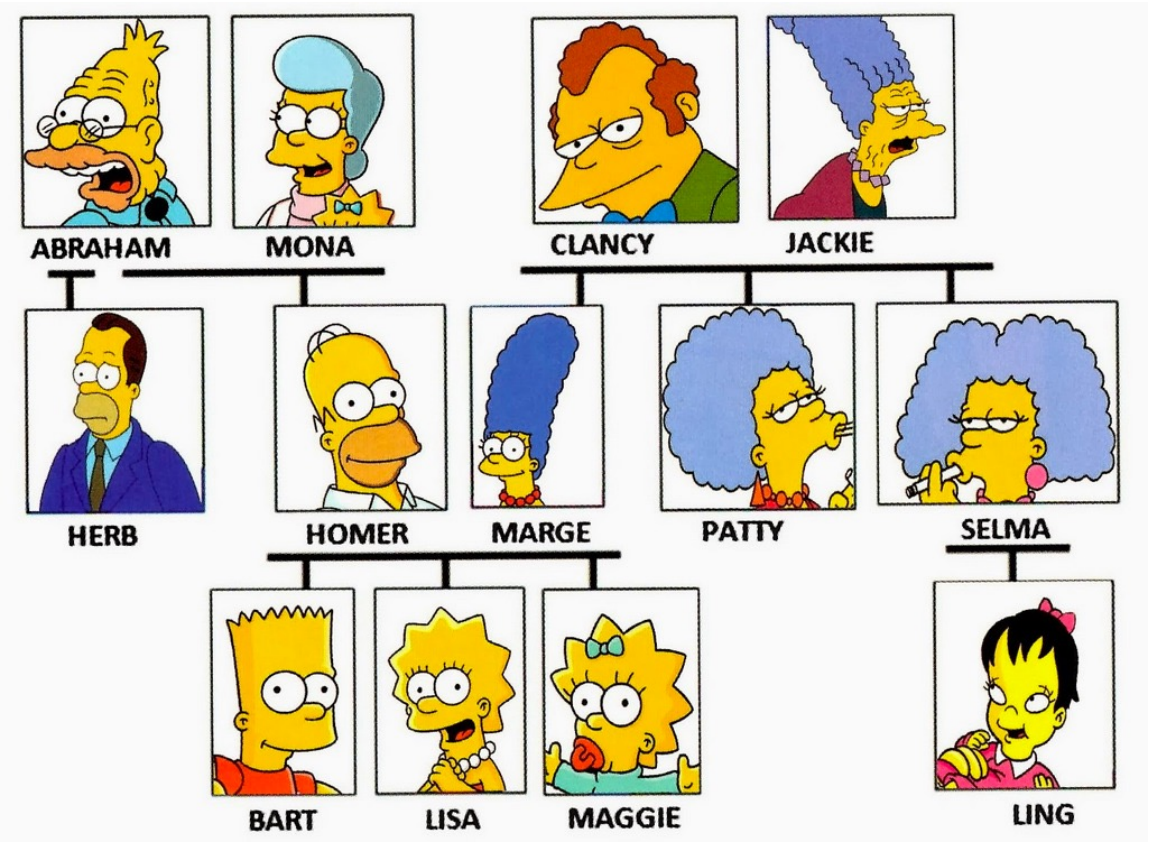
Lista de listas rectangular

i

j

	i-1, j-1	i-1, j	i-1, j+1	
	i, j-1	i,j	i, j+1	
	i+1, j-1	i, j+1	i+1, j+1	

Problema #3


ABRAHAM
MONA
CLANCY
JACKIE
HERB
HOMER
MARGE
PATTY
SELMA
BART
LISA
MAGGIE
LING

Problema #3

Estás realizando tu práctica en `GeoTree`, un emprendimiento UC que usa Python para armar árboles genealógicos. Sin embargo, están teniendo problemas en la implementación, por lo que te piden a ti, flamante alumno de Introducción a la Programación, que los ayudes a terminar su programa.

Hasta ahora, lo único que tienen es una representación con *listas* del árbol de una familia, y la declaración de las funciones que vas a necesitar. El árbol está representado por una lista donde, en cada elemento se guarda un *string* con el nombre del miembro de la familia (no hay *strings* repetidos), seguido de una *lista* de las posiciones de sus hijos dentro de la lista. Puedes ver un ejemplo de la lista en el código más abajo. Las funciones que necesitan que implementes son las siguientes:

(a) **(30 puntos)** `hijos(a,n)`: recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una *lista* vacía.

(b) **(30 puntos)** `padres(a,n)`: recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una *lista* vacía.

A continuación, se encuentra el código con el ejemplo de la familia Simpson y el *output* esperado escrito en los comentarios. Tu deber es completar las funciones `hijos(a,n)` y `padres(a,n)`, para que los `print` del final del código impriman lo solicitado. Ten en cuenta que no es necesario que la lista de los nombres salgan exactamente en el orden del *output* esperado.

```python
def hijos(a, n):
    #Completar

def padres(a, n):
    #Completar

simp = [
    ["Abraham",[1,2]],     #0
    ["Herb",[]],           #1
    ["Homer",[3,4,5]],     #2
    ["Bart",[]],           #3
    ["Maggie",[]],         #4
    ["Lisa",[]],           #5
    ["Marge",[3,4,5]],     #6
    ["Mona",[2]],          #7
    ["Clancy",[6,10,11]], #8
    ["Jackie",[6,10,11]], #9
    ["Selma",[12]],        #10
    ["Patty",[]],          #11
    ["Ling",[]]            #12
    ]

print(hijos(simp,'Marge'))       #['Bart', 'Maggie', 'Lisa']
print(hijos(simp,'Clancy'))      #['Marge', 'Patty', 'Selma']
print(hijos(simp, 'Maggie'))     #[]
print(padres(simp, 'Maggie'))    #['Homer', 'Marge']
print(padres(simp, 'Homer'))     #['Abraham', 'Mona']
print(padres(simp, 'Ling'))      #['Selma']
print(padres(simp, 'Abraham'))   #[]
```

Resumen de Listas

- **Hoy vimos**:
- `Lista de listas`
- `M= [[0 for x in range(cols)] for x in range(filas)]`
- Casillero en fila i, columna j: `M[i][j]`
- Número filas: `len(M)`
- Número columnas: `len(M[0])` o `len(M[i])`