10 – LISTAS DE LISTAS

Jorge Muñoz

IIC1103 – Introducción a la Programación

LISTAS DE LISTAS (nada nuevo)

```python
#Lista de ints (6 elementos)
a = [5, -1, 4, 9, 23, -3]

#Lista de str (4 elementos)
b = ["arbol","barco","casa","dado"]

#Lista de mix de basicos (6 elementos)
c = [5, "hola", 6, 7, True, 6.23]
```

```python
# Lista de listas (4 elementos)
# Cada elemento es una lista [str,float] con el nombre y nota de un estudiante
d = [["Alice",5.5],["Bob",4.3],["Charlie",6.2],["David",6.0]]
```

```python
# Lista de listas (4 elementos)
# Cada elemento es una lista [str,float] con el nombre y nota de un estudiante
d = [["Alice",5.5],["Bob",4.3],["Charlie",6.2],["David",6.0]]


#Lista de mix de tipos (5 elementos)
#No tienen por que ser todos los elementos del mismo tipo
e = [7,"adios",["precio",1100], True, 5.31]

#Lista de list (3 elementos)
#No tienen por que ser todas las listas iguales
f = [["arbol","abeja"],["barco","banco","bar"],["caracol"]]


# Lista (3 elementos)
# Las listas pueden tener listas, y pueden tener listas, ...
g = [["Calculo",[4.2,5.5]],["Quimica",[4.5]],["Algebra",[5.3,3.2,4.4,6.8]]]


# Misma Lista (3 elementos)
# Python te deja escribirlo separado para que lo puedas leer mejor
# pero es exactamente lo mismo

h = [["Calculo",[4.2,5.5]],
     ["Quimica",[4.5]],
     ["Algebra",[5.3,3.2,4.4,6.8]]]
```

CREANDO LISTAS DE LISTAS (nada nuevo)

```python
#Crear lista de nombres (list de str)
nombres = []

num = int(input("Cuantos nombres? "))
for i in range(0,num):

    nombre = input("Nombre? ")
    nombres.append(nombre)

print(nombres)
```

```
Cuantos nombres? 3
Nombre? Alicia
Nombre? Bernardo
Nombre? Carlos
['Alicia', 'Bernardo', 'Carlos']
```

```
#Crear lista de nombres (list de listas [str,float])
estudiantes = []

num = int(input("Numero estudiantes? "))
for i in range(0,num):

    nombre = input("Nombre? ")
    nota = float(input("Nota? "))

    est = [nombre,nota]
    estudiantes.append(est)

print(estudiantes)
```

```
Numero estudiantes? 3
Nombre? Alicia
Nota? 6.8
Nombre? Bernardo
Nota? 3.5
Nombre? Carlos
Nota? 5.5
[['Alicia', 6.8], ['Bernardo', 3.5], ['Carlos', 5.5]]
```

ENCADENADOS (nada nuevo)

```
compra = [["Aceite",3500],["Pasta",1300],["Arroz",980],["Pollo",2340]]

# Producto en la posicion 2 de la lista de la compra
prod = compra[2]
# Su Nombre y precio
nombre = prod[0]
precio = prod[1]


# Puedes encadenarlo para ir mas directo
# compra[2] es una lista, no? pues a una lista le puedes
# poner el [] pegado para acceder a un elemento.
# Es exactamente LO MISMO de siempre
nombre = compra[2][0]
precio = compra[2][1]



# Lo mismo con listas de listas de listas, ...
qs = [["USA",["MIT","Harvard","Stanford"]],
        ["UK",["Oxford","Cambridge"]],
        ["Japan",["Todai","Sokendai"]]]

#El nombre de la primera universidad de USA
top = qs[0][1][0]

#La larga
usa = qs[0] #Lista [str,list]
unis = usa[1] # Lista de str con las unis
top = unis[0] # str con la top uni (pos 0)
```

PPA

```
def ppa(es):
```



```
es = [["Alice",[6.6,5.4,6.0]],
      ["Barto",[4.0,4.0]],
      ["Carlo",[5.5,6.2,6.9,7.0]],
      ["Peter",[5.5,4.3]]]
```

```
[['Alice', 6.0], ['Barto', 4.0], ['Carlo', 6.4], ['Peter', 4.9]]
```

```python
def ppa(es):

    #Creo una lista donde pondre listas ([nombre,ppa])
    datos = []

    #Miro estudiante a estudiante
    for e in es:
        nombre = e[0]
        notas = e[1]

        #Calculo el promedio(ppa)
        suma = 0
        for n in notas:
            suma += n
        ppa = suma / len(notas)

        #Creo una lista
        dato = [nombre,ppa]

        #La agrego a mi lista de datos
        datos.append(dato)

    #Retornar
    return datos
```

```python
es = [["Alice",[6.6,5.4,6.0]],
      ["Barto",[4.0,4.0]],
      ["Carlo",[5.5,6.2,6.9,7.0]],
      ["Peter",[5.5,4.3]]]
```

```
[['Alice', 6.0], ['Barto', 4.0], ['Carlo', 6.4], ['Peter', 4.9]]
```

EUROVISION



Situación Real

openup

EUROVISION
SONG CONTEST
ROTTERDAM 2020

Situación Real

12	NORTH MACEDONIA	91
6	SWEDEN	75
5	ITALY	74
8	AZERBAIJAN	69
1	RUSSIA	66
	THE NETHERLANDS	65
2	AUSTRALIA	63
10	SWITZERLAND	60
	CZECH REPUBLIC	49
	MALTA	41
	DENMARK	36
7	CYPRUS	34
	ALBANIA	33
	SERBIA	25
3	FRANCE	22
4	GREECE	22
	SLOVENIA	18
	ICELAND	11
	NORWAY	6
	ESTONIA	5
	BELARUS	2
	UNITED KINGDOM	2
	SAN MARINO	1
	GERMANY	0
	ISRAEL	0
	SPAIN	0



12 10 8 7 6 5 4 3 2 1

15 OF 41 COUNTRIES VOTING

```python
# Recibe una lista de votos,
# donde cada voto es una lista [str,str,int]
# con el nombre del pais que ha votado,
# a quien ha votado y cuantos puntos,
# y retorna un str con el ganador de Eurovision

def eurovision(vs):
```



```python
votos = [["Spain","Italy",10],
         ["Spain","Ireland",12],
         ["France","Ireland",8],
         ["France","Spain",1],
         ["Italy","Spain",10],
         ["Italy","Ireland",12]]
```

Ireland

CONSEJO

"En vez de hacerlo directamente, a veces vale la pena descomponer: 1) ir procesando y guardando datos en algún sitio, 2) y luego usar eso para contestar la pregunta fácilmente"

```python
# Recibe una lista de votos,
# donde cada voto es una lista [str,str,int]
# con el nombre del pais que ha votado,
# a quien ha votado y cuantos puntos,
# y retorna un str con el ganador de Eurovision

def eurovision(vs):
```

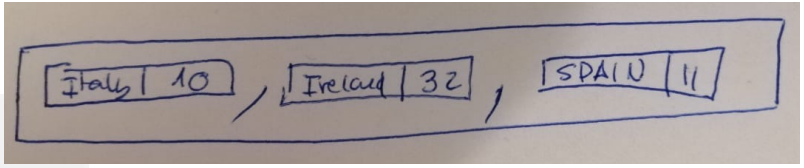


```python
votos = [["Spain","Italy",10],
         ["Spain","Ireland",12],
         ["France","Ireland",8],
         ["France","Spain",1],
         ["Italy","Spain",10],
         ["Italy","Ireland",12]]
```

Ireland

```python
# Recibe una lista de votos,
# donde cada voto es una lista [str,str,int]
# con el nombre del pais que ha votado,
# a quien ha votado y cuantos puntos,
# y retorna un str con el ganador de Eurovision

def eurovision(vs):
    # Voy a crear una lista de listas [pais,puntos]
    # donde voy a ir sumando los puntos que vaya sacando
    # cada pais
    puntos = []
    ...
```




```python
votos = [["Spain","Italy",10],
         ["Spain","Ireland",12],
         ["France","Ireland",8],
         ["France","Spain",1],
         ["Italy","Spain",10],
         ["Italy","Ireland",12]]
```

Ireland

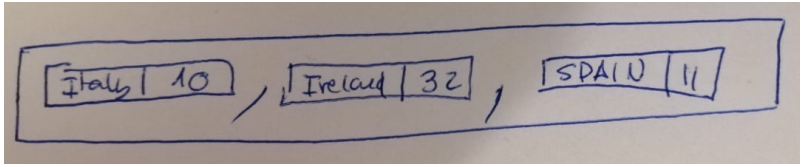
```python
# Recibe una lista de votos,
# donde cada voto es una lista [str,str,int]
# con el nombre del pais que ha votado,
# a quien ha votado y cuantos puntos,
# y retorna un str con el ganador de Eurovision

def eurovision(vs):
    # Voy a crear una lista de listas [pais,puntos]
    # donde voy a ir sumando los puntos que vaya sacando
    # cada pais
    puntos = []

    #Para cada voto
    for v in vs:

        #Solo me interesa el pais que recibe los puntos y los puntos
        pais = v[1]
        pts = v[2]

        #Voy a buscar si ya tengo ese pais en mi lista de puntos
        #Y si lo tengo, le aumento los puntos
        esta = False
        for p in puntos:
            if p[0] == pais:
                esta = True
                p[1] += pts
```




```python
votos = [["Spain","Italy",10],
         ["Spain","Ireland",12],
         ["France","Ireland",8],
         ["France","Spain",1],
         ["Italy","Spain",10],
         ["Italy","Ireland",12]]
```

...

Ireland

```python
# Recibe una lista de votos,
# donde cada voto es una lista [str,str,int]
# con el nombre del pais que ha votado,
# a quien ha votado y cuantos puntos,
# y retorna un str con el ganador de Eurovision

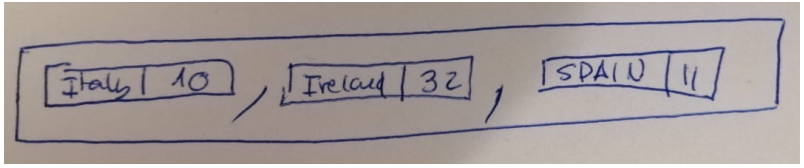
def eurovision(vs):
    # Voy a crear una lista de listas [pais,puntos]
    # donde voy a ir sumando los puntos que vaya sacando
    # cada pais
    puntos = []

    #Para cada voto
    for v in vs:

        #Solo me interesa el pais que recibe los puntos y los puntos
        pais = v[1]
        pts = v[2]

        #Voy a buscar si ya tengo ese pais en mi lista de puntos
        #Y si lo tengo, le aumento los puntos
        esta = False
        for p in puntos:
            if p[0] == pais:
                esta = True
                p[1] += pts

        #Si no esta, me toca crearlo y ponerle los puntos
        if not esta:
            x = [pais,pts]
            puntos.append(x)
```




```python
votos = [["Spain","Italy",10],
         ["Spain","Ireland",12],
         ["France","Ireland",8],
         ["France","Spain",1],
         ["Italy","Spain",10],
         ["Italy","Ireland",12]]
```

...

Ireland

```python
# Recibe una lista de votos,
# donde cada voto es una lista [str,str,int]
# con el nombre del pais que ha votado,
# a quien ha votado y cuantos puntos,
# y retorna un str con el ganador de Eurovision

def eurovision(vs):
    # Voy a crear una lista de listas [pais,puntos]
    # donde voy a ir sumando los puntos que vaya sacando
    # cada pais
    puntos = []

    #Para cada voto
    for v in vs:

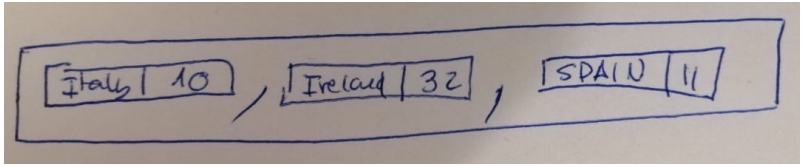
        #Solo me interesa el pais que recibe los puntos y los puntos
        pais = v[1]
        pts = v[2]

        #Voy a buscar si ya tengo ese pais en mi lista de puntos
        #Y si lo tengo, le aumento los puntos
        esta = False
        for p in puntos:
            if p[0] == pais:
                esta = True
                p[1] += pts

        #Si no esta, me toca crearlo y ponerle los puntos
        if not esta:
            x = [pais,pts]
            puntos.append(x)

    #Aqui tendre en puntos todos los paises que han recibido puntos
    # y los puntos que tienen. Solo tengo que buscar quien tiene mas
    maxip = -1
    maxin = ""
    for p in puntos:
        if p[1] > maxip:
            maxip = p[1]
            maxin = p[0]

    return maxin
```




```python
votos = [["Spain","Italy",10],
         ["Spain","Ireland",12],
         ["France","Ireland",8],
         ["France","Spain",1],
         ["Italy","Spain",10],
         ["Italy","Ireland",12]]
```

Ireland


© BBC





Bibliografía &
Investigación

GEOTREE

IIC1103 – Introducción a la Programación
2018 - 2

Interrogación 2

Pregunta 1

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.

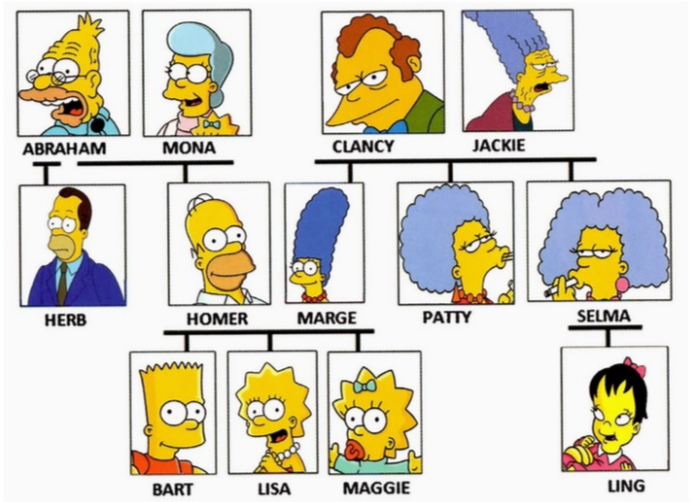
Hasta ahora, lo único que tienen es una representación con *listas* del árbol de una familia, y la declaración de las funciones que vas a necesitar. El árbol está representado por una lista donde, en cada elemento se guarda un *string* con el nombre del miembro de la familia (no hay *strings* repetidos), 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.

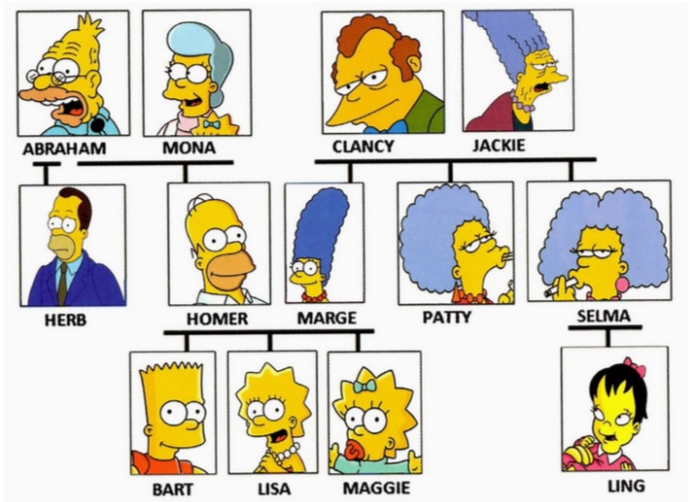
(a) **(30 puntos)** `hijos(a,n)`: recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una *lista* vacía.




```
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'))     #[]
```

(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.




```python
def hijos(a,n):
    res = []

    #Recorrer hasta encontrar la persona (n)
    for p in a:
        nombre = p[0]

        #Si es la persona que busco
        if nombre == n:
            #Lista de posiciones de los hijos
            phs = p[1]

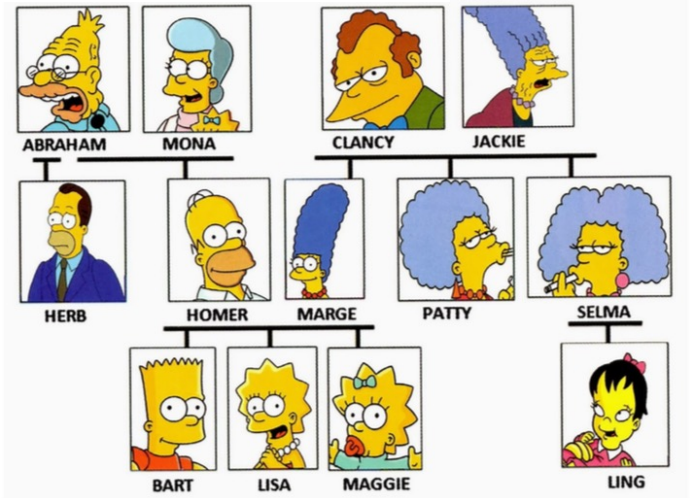
            #Por cada pos de cada hijo
            for ph in phs:
                nombre = a[ph][0]
                res.append(nombre)

    return res
```

```python
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
    ]
```

```python
print(hijos(simp,'Marge'))      #['Bart', 'Maggie', 'Lisa']
print(hijos(simp,'Clancy'))     #['Marge', 'Patty', 'Selma']
print(hijos(simp, 'Maggie'))    #[]
```

(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.




```
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(padres(simp, 'Maggie'))     #['Homer', 'Marge']
print(padres(simp, 'Homer'))      #['Abraham', 'Mona']
print(padres(simp, 'Ling'))       #['Selma']
print(padres(simp, 'Abraham'))    #[]
```

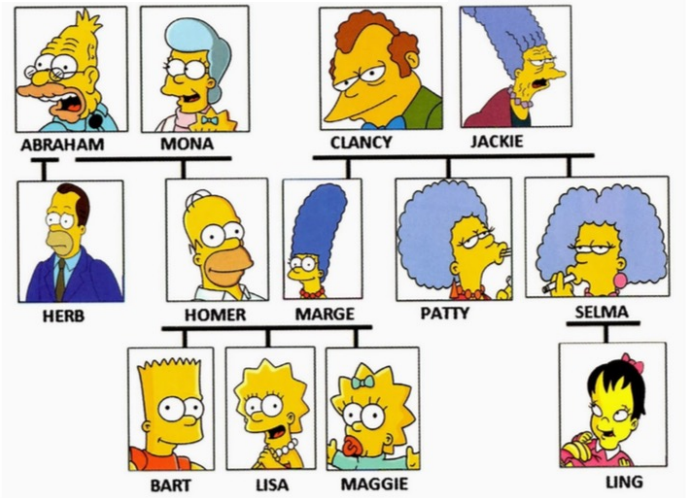
(b) **(30 puntos)** `padres(a,n)`: recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una *lista* vacía.

```python
def padres(a,n):
    #Buscar la posicion de la persona
    for i in range(len(a)):
        nombre = a[i][0]
        if nombre == n:
            pos = i

    #Buscar gente que tiene pos en sus hijos
    res = []
    for p in a:
        nom = p[0]
        phjs = p[1]
        if pos in phjs:
            res.append(nom)

    return res
```

```python
print(padres(simp, 'Maggie'))    #['Homer', 'Marge']
print(padres(simp, 'Homer'))     #['Abraham', 'Mona']
print(padres(simp, 'Ling'))      #['Selma']
print(padres(simp, 'Abraham'))   #[]
```




```python
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
    ]
```

MAPAS (nada nuevo)

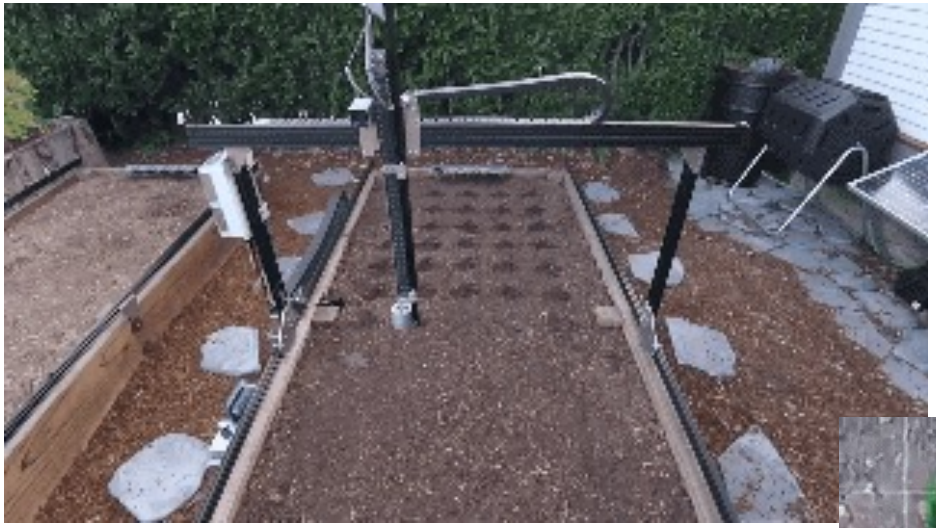
INTERESES

“Agronomia”

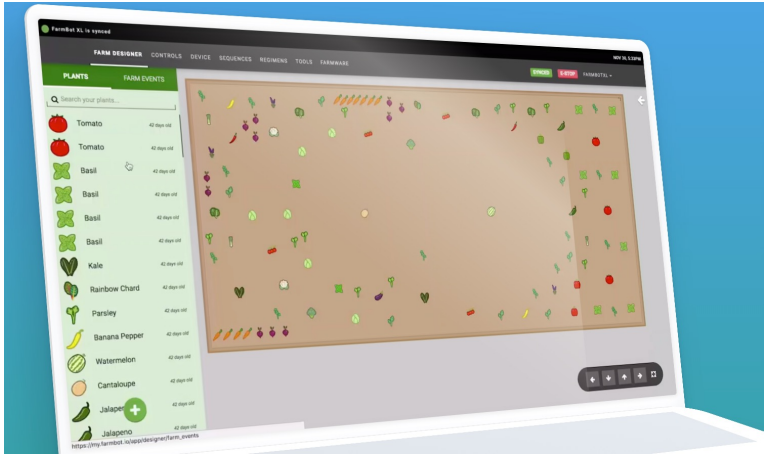
“aplicaciones sobre temas agronómicos, por ejemplo algún robot que deba cultivar y cosechar diversas plantas en tales épocas del año, etc...”

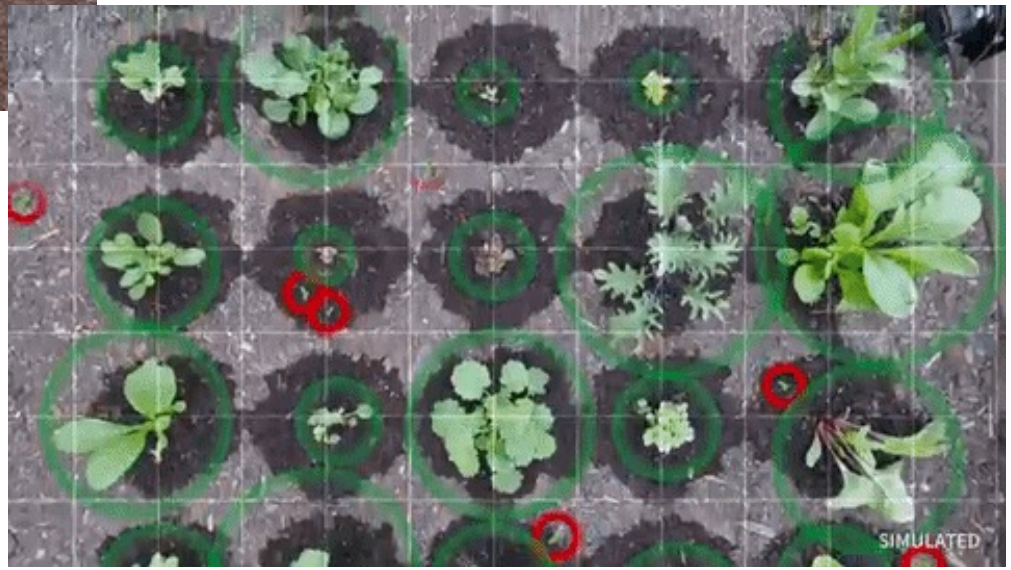
“cuidado del medio ambiente y recursos hídricos”

Situación Real

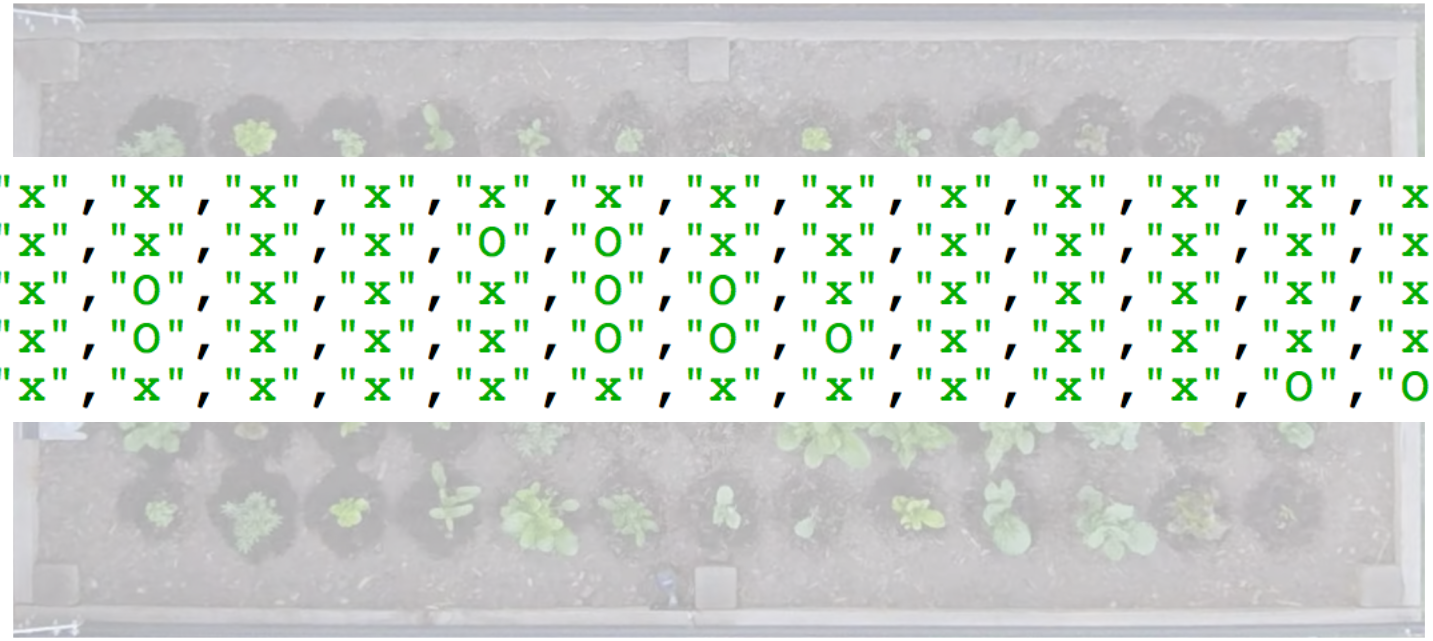


FarmBot









```
g = [["x","x","x","x","x","x","x","x","x","x","x","x","x"],
     ["x","x","x","x","O","O","x","x","x","x","x","x","x"],
     ["x","O","x","x","x","O","O","x","x","x","x","x","x"],
     ["x","O","x","x","x","O","O","O","x","x","x","x","x"],
     ["x","x","x","x","x","x","x","x","x","x","x","O","O"]]
```

- FarmBot
  - https://farm.bot/
  - https://www.youtube.com/watch?v=60htrqei_U0
  - https://developer.farm.bot/docs
  - https://github.com/openfarmcc

ZIPPEDI

Situación Real





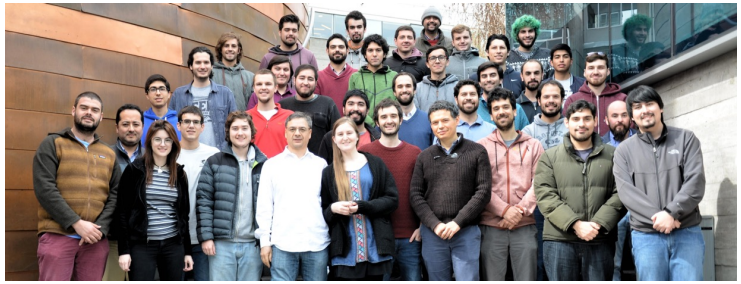
ÁLVARO SOTO

Profesor Asociado

Especialidad: Apredizaje de Máquina, Big Data, Robótica Cognitiva, Reconocimiento Visual.

Departamento de Ciencia de la Computación

asoto@ing.puc.cl
+56223544440



IA Lab

emol. Tecnología

Santiago: Jueves 05 de abril del 2018 | Actualizado 09:45

Noticias | Economía | Deportes | Espectáculos | Tendencias | Autos | Servicios | [O] | [tV] | 360°

Chile | Mundo | Tecnología | Educación | Documentos | Multimedia

Buscar | emol.com

Zippedi: Cómo es el primer robot chileno que trabaja en supermercados

Gracias al uso de Inteligencia Artificial, el robot revisa precios y el orden de los productos en los estantes de locales comerciales.

10 de Enero de 2018 | 13:29 | Emol

2



Zippedi

SANTIAGO.- Un grupo liderado por Luis Vera y Ariel Schilkrut, fundadores de Zippedi, desarrolló el primer robot chileno que cuenta con Inteligencia Artificial y que tiene el objetivo de **verificar que los precios y orden de los productos en supermercados no presenten errores.**

Luego de tres años de investigación, el grupo de científicos y alumnos de postgrado de Ciencia de la Computación de la Universidad Católica fue capaz de desarrollar esta innovación a la cual llamaron Zippedi. Junto al apoyo de Corfo, Conicyt y distintas cadenas de retail del país **lograron crear el robot íntegramente en Chile.**

AHORA SE DEBATE



Hacienda aprieta el cinturón y hará ajuste fiscal por US$500 millones. ¿Cómo lo ves?

176



¿A qué crees que se debe la crisis que atraviesa la Municipalidad de Viña del Mar y Virginia Reginato?

228




AKBAR
Dilmah
GOLD
CRISTAL

Con fecha 21-06-2017 22:42 se publico el siguiente aviso en el curso IIC1103 s.6 Introducción a la Programación:

Estimados alumnos,

El examen del día sábado 24 de junio será a las 9:00 hrs. y durará 4 horas.

Se les recomienda que lleven un snack por la duración de éste.

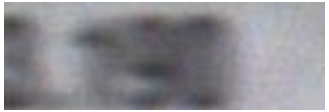
Además mencionarles que el grupo de Inteligencia de Máquina de la PUC busca ayuda para generar datos que ayuden a mejorar las capacidades cognitivas del robot que se encuentran desarrollando. Pueden ayudarlos rotulando en la pagina:

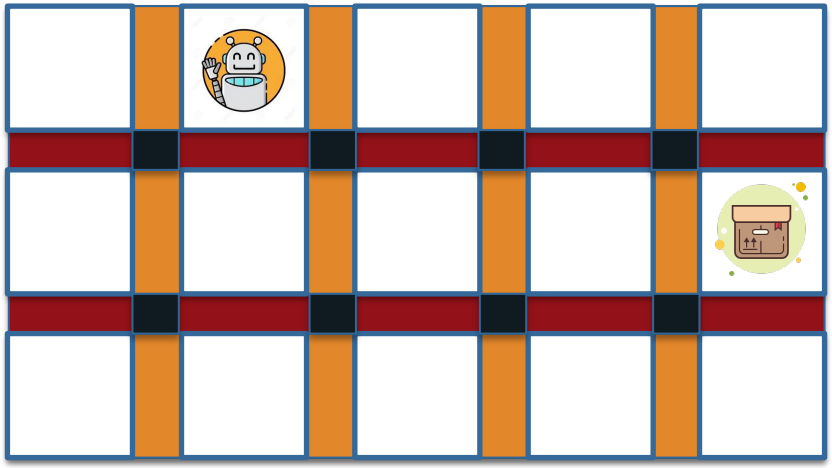
http://grima-labeling.ing.puc.cl











```
[["_","R","_","_","_"],
 ["_","_","_","_","P"],
 ["_","_","_","_","_"]]
```

5

1 seg pasar por naranja y 2 seg por roja

Función que recibe un mapa (list de list) con la situación del robot y el producto, y retorna un int con el tiempo

```
def zippedi(m):

    nfilas = len(m)
    ncols = len(m[0])
    #Num de elementos de cualquier fila
    #por ejemplo, la 0

    #Buscar donde esta el robot y el producto
    for f in range(0,nfilas):
        for c in range(0,ncols):
            if m[f][c] == "R":
                rf = f
                rc = c
            elif m[f][c] == "P":
                pf = f
                pc = c

    #Naranjas
    nar = rc-pc
    #Puede ser pos o neg dependiendo de si el
    #robot esta a la izq o derecha del produc
    #Si es neg lo hacemos pos (mult por -1)
    if nar < 0:
        nar =  nar * -1

    #Rojas
    nro = rf - pf
    if nro < 0:
        nro = nro * -1

    tiempo = nar*1 + nro*2
    return tiempo
```

```
[[" ","R"," "," "," "],
 [" "," "," "," ","P"],
 [" "," "," "," "," "]]
```

BUSCAMINAS


Easy
10
013

```
# Asumir no esta en el borde
def minas(m,f,c):
```



```
    return num
```

```
[["_","X","X","_"],
 ["_","_","_","_"],
 ["X","_","X","X"],
 ["_","_","X","X"],
 ["_","_","_","_"]]
```

```
F: 2 C: 1 Minas: 3
F: 1 C: 2 Minas: 4
```

```
# Asumir no esta en el borde
def minas(m,f,c):
    num = 0
    if m[f-1][c-1] == "X":
        num += 1
    if m[f-1][c] == "X":
        num += 1
    if m[f-1][c+1] == "X":
        num += 1
    if m[f][c-1] == "X":
        num += 1
    if m[f][c+1] == "X":
        num += 1
    if m[f+1][c-1] == "X":
        num += 1
    if m[f+1][c] == "X":
        num += 1
    if m[f+1][c+1] == "X":
        num += 1

    return num
```

```
[[" ","X","X"," "],
 [" "," "," "," "],
 ["X"," ","X","X"],
 [" "," ","X","X"],
 [" "," "," "," "]]
```

```
F: 2 C: 1 Minas: 3
F: 1 C: 2 Minas: 4
```

Situación Real

¿Algo menos manual?

(Imaginar que en vez solo las que tocan, te pidieran mirar las que están distancia 2 o 3)

```python
# Asumir no esta en el borde
def minas(m,f,c):
    num = 0

    for i in range(-1,2):
        for j in range(-1,2):

            if m[f+i][c+j] == "X":
                num += 1
    return num
```

```
[["_","X","X","_"],
 ["_","_","_","_"],
 ["X","_","X","X"],
 ["_","_","X","X"],
 ["_","_","_","_"]]

F: 2 C: 1 Minas: 3
F: 1 C: 2 Minas: 4
```

¿Y si está en un borde?

```
m = [["_","X","X","_"],
     ["_","_","_","_"],
     ["X","_","X","X"],
     ["_","_","X","X"],
     ["_","_","_","_"]]
```

```
r3 = minas(m,4,2)
print("F:",4,"C:",2,"Minas:",r3)
```

```
e/publico/codigos/10-ListasDeListas/buscaminas.py", line 7, in minas
    if m[f+i][c+j] == "X":
IndexError: list index out of range
```

```
r0 = minas(m,0,1)
print("F:",0,"C:",1,"Minas:",r0)
```

```
F: 0 C: 1 Minas: 2
```

```
def minas(m,f,c):

    nf = len(m)
    #Num cols son los elems de cualquier fila
    #por ejemplo la 0
    nc = len(m[0])

    num = 0

    for i in range(-1,2):
        for j in range(-1,2):

            fdentro = (f+i >= 0 and f+i < nf)
            cdentro = (c+j >= 0 and c+j < nc)

            if fdentro and cdentro:
                if m[f+i][c+j] == "X":
                    num += 1
    return num
```

```
[[" ","X","X"," "],
 [" "," "," "," "],
 ["X"," ","X","X"],
 [" "," ","X","X"],
 [" "," "," "," "]]
```

```
F: 0 C: 1 Minas: 2
F: 2 C: 1 Minas: 3
F: 1 C: 2 Minas: 4
F: 4 C: 2 Minas: 2
```

MUTABILIDAD (nada nuevo)

Strings

```
a = "Mal 3scrito"
a[4] = "e"
print(a)
```

```
    a[4] = "e"
TypeError: 'str' object does not support item assignment
```

Listas

```
a = ["Mal", "3scrito"]
a[1] = "escrito"
print(a)
```

```
['Mal', 'escrito']
```

```
a = [24, 50]
a[1] += 10
print(a)
```

```
[24, 60]
```

Listas de listas

```
a = [
    "Algo",
    [1, 2, 3],
    [4, 5, ["Hola", "Mundo"]]
    ]
a[0] = "Nada"
a[1][1] = 99
a[2][2][0] = "Adios"
print(a)
```



Listas de listas

```
a = [
    "Algo",
    [1, 2, 3],
    [4, 5, ["Hola", "Mundo"]]
    ]
a[0] = "Nada"
a[1][1] = 99
a[2][2][0] = "Adios"
print(a)
```

```
['Nada', [1, 99, 3], [4, 5, ['Adios', 'Mundo']]]
```

Ejemplo: contar letras de una frase

```
lista_letra_aparicion = [
    [letra_1, cant_1],
    [letra_2, cant_2],
    ...
]
```

```python
def contar_letras(frase):
    lista_letra_aparicion = []

    # Por cada letra en la frase
    for c in frase:
        encontrada = False

        # Buscamos si la letra ya está en la lista
        [redacted]

        # Si no se encontró la letra la agregamos
        if not encontrada:
            lista_letra_aparicion.append([c, 1])

    return lista_letra_aparicion

s = "hello world"
print(contar_letras(s))
```

Ejemplo: contar letras de una frase

```
lista_letra_aparicion = [
    [letra_1, cant_1],
    [letra_2, cant_2],
    ...
]
```

```python
def contar_letras(frase):
    lista_letra_aparicion = []

    # Por cada letra en la frase
    for c in frase:
        encontrada = False

        # Buscamos si la letra ya está en la lista
        for item in lista_letra_aparicion:
            letra = item[0]
            if letra == c:
                item[1] += 1
                encontrada = True
                break

        # Si no se encontró la letra la agregamos
        if not encontrada:
            lista_letra_aparicion.append([c, 1])

    return lista_letra_aparicion

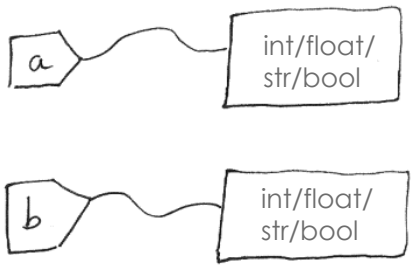
s = "hello world"
print(contar_letras(s))
```

```
[['h', 1], ['e', 1], ['l', 3], ['o', 2],
 [' ', 1], ['w', 1], ['r', 1], ['d', 1]]
```

COPIANDO LISTAS DE LISTAS

Otros

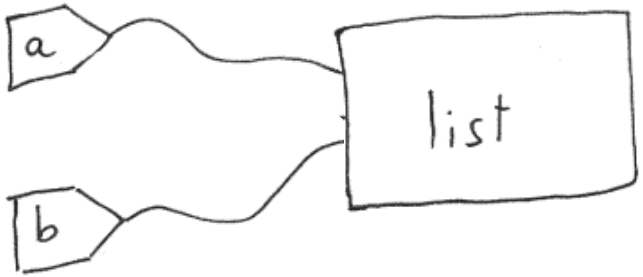
```
a = 10
b = a
b = 5
print(a)    10
print(b)    5
```



a y b referencian
elementos distintos

Listas

```
a = [0]
b = a
b[0] += 10
b.append("hola")
print(a)    [10, 'hola']
print(b)    [10, 'hola']
```



a y b referencian
a la misma lista

Listas de listas

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

a_copy = a

a_copy.append([10, 11, 12])
print(a)
print(a_copy)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

```
a_copy[0][0] = 99
print(a)
print(a_copy)
```

```
[[99, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
[[99, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

Ambas referencian a la misma lista

Un cambio en una afecta a la otra

Listas de listas

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

a_copy = []
for f in a:
    a_copy.append(f)

a_copy.append([10, 11, 12])
print(a)
print(a_copy)

a_copy[0][0] = 99
print(a)
print(a_copy)
```





Listas de listas

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

a_copy = []
for f in a:
    a_copy.append(f)

a_copy.append([10, 11, 12])
print(a)
print(a_copy)

a_copy[0][0] = 99
print(a)
print(a_copy)
```

Logramos crear una copia de la lista externa, pero las interiores siguen siendo las mismas

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

```
[[99, 2, 3], [4, 5, 6], [7, 8, 9]]
[[99, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

Listas de listas

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

a_copy = []
for f in a:
    f_copy = []
    for e in f:
        f_copy.append(e)
    a_copy.append(f_copy)

a_copy.append([10, 11, 12])
print(a)
print(a_copy)

a_copy[0][0] = 99
print(a)
print(a_copy)
```

Tenemos que también copiar todas las listas interiores

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[99, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```