- Implementación de la clase Conjunto del tema 8.
- Problema de examen:

Se dispone de un fichero de texto de nombre "resultadosprimera.txt" con los resultados de todos los partidos de fútbol de primera división disputados en una determinada temporada. Cada línea del fichero almacena la información de un partido y contiene, separados por almohadillas, los siguientes datos: el nombre del equipo local, el número de goles que ha marcado el equipo local, el nombre del equipo visitante y el número de goles que ha marcado el equipo visitante. Por ejemplo:

```
Celta de Vigo#0#Málaga#1
      Sevilla#2#Getafe#1
      Mallorca#2#Espanyol#1
      Espanyol#3#Mallorca#2
      Granada#2#Rayo Vallecano#0
      Getafe#1#Sevilla#1
```

Implementar en Python:

a) (2.5 puntos) Una función o procedimiento leer_datos_partidos que lea los datos del fichero y los almacene en una matriz (lista de listas) con tantas filas como líneas contiene el fichero y con 4 columnas, una para cada uno de los datos que se describen arriba.

b) (2.5 puntos) Una función o procedimiento resumen_datos_equipo que, dada una matriz como la que se describe en el primer apartado y el nombre de un equipo, devuelva una lista con el número de goles a favor (marcados), el número de goles en contra (encajados) y los puntos totales que ha obtenido. Los equipos obtienen tres puntos por cada partido ganado, un punto por cada empate y ningún punto por los partidos perdidos. Por ejemplo, dados los datos de arriba y el equipo 'Celta de Vigo', la función deberá devolver [0,1,0].

c) (2.5 puntos) Una función o procedimiento mejor_clasificado que, dada una matriz como la que se describe en el primer apartado y una lista con los nombres de todos los equipos, devuelva el nombre del equipo que encabeza la clasificación. Un equipo está mejor clasificado que otro si tiene más puntos. En caso de empate a puntos, para simplificar supondremos que el empate se resuelve eligiendo el equipo que tenga una mayor diferencia entre goles a favor y en contra. Por ejemplo, dados los datos de arriba y una lista con todos los equipos, la función deberá devolver 'Sevilla' (sus datos son [3,2,4]). Importante: La función/procedimiento deberá llamar obligatoriamente a resumen_datos_equipo.

- Implementación de una clase Board (Tablero) para utilizarla en el juego del "buscaminas" de la práctica 5.

```
# problema 1.a
# Crear una lista de float a partir de una cadena con dígitos, puntos y
espacios
def crear_lista_de_cadena(cadena):
    lista = []
    aux = ''
    ant = ' '
    no_valido = False
    for c in cadena:
        # es dígito o punto y no hay uno ya en aux
        if (c >= '0' and c <= '9') or (c == '.' and '.' not in aux):
            aux += c
        elif c == '.':  # es punto (y hay uno ya en aux)
            no_valido = True
            break
        elif c == ' ' and ant != ' ':  # es espacio después de un
caracter
            lista.append(float(aux))
            aux = ''
        elif c != ' ':  # es un caracter no válido
            no_valido = True
            break
        ant = c

    # si al terminar queda algo en aux (no hay espacio al final)
    if no_valido==False and len(aux)>0:
        lista.append(float(aux))

    if no_valido==False:
        return lista
    else:
        return None

# problema 1.b
# Crear una matriz cuadrada de float pidiendo la dimensión y las filas
como cadenas,
# comprobando que son adecuadas
def leer_matriz_por_filas():
    dimension = 1
    while dimension<2 or dimension>100:
        dimension = int(input('¿Dimensión de la matriz? (entre 2 y 100)
'))
    matriz=[]
    while len(matriz)<dimension:
        línea = input('¿Números flotantes para la fila? (separados por
espacios) ')
        fila = crear_lista_de_cadena(línea)
        if fila==None or len(fila)!=dimension:
            print('Error, carácter no válido o lista con
demasiados/pocos elementos.')
        else:
            print(fila)
            matriz.append(fila)
```

```
    return matriz

# problema 2.a
# Leer de un fichero de texto con líneas del tipo
# "Sinatra, F.#1960#Ocean's Eleven#Danny Ocean\n"
# a una matriz
def split_a_mano(cadena):
    resultado=[]
    i=0
    dato=''
    for caracter in cadena:
        if caracter!='#' and caracter!='\n':
            dato+=caracter
        else:
            resultado.append(dato)
            dato=''
    return resultado

def leer_datos():
    matriz=[]
    fichero = open('actorsEXA.txt', 'r')
    for linea in fichero:
        fila=split_a_mano(linea)
        fila[C_AÑO] = int(fila[C_AÑO])
        matriz.append(fila)
    fichero.close()
    return matriz

# problema 2.b
# Crear una lista de pares (tuplas) de actores a partir de la matriz y
del año y el
# título de una peli
def pelicula_pares_de_actores(matriz,año,peli):
    actores = []
    for fila in matriz:
        if fila[C_AÑO]==año and fila[C_PELI]==peli:
            actores.append(fila[C_ACTOR])

    pares = []
    for i in range(len(actores)):
        for j in range(i+1,len(actores)):
            pares.append((actores[i],actores[j]))

    return pares

# problema 2.c
# Contar las veces que aparecen juntos en una peli dos actores, usando
la función
# anterior
def actores_actuan_juntos_veces(matriz,actor1,actor2):
    veces = 0
    for fila in matriz:
        if fila[C_ACTOR]==actor1:
            pares =
pelicula_pares_de_actores(matriz,fila[C_AÑO],fila[C_PELI])
            for par in pares:
                if actor1 in par and actor2 in par:
                # también sirve:
```

```
                # if (actor1,actor2)==par or (actor2,actor1)==par:
                # if par[0]==actor2 or par[1]==actor2:
                    veces += 1
                    break  # una vez encontrado el par no hace falta
seguir con esa peli
    return veces

Problema de examen:
```

Este es el de Julio de 2014

Se dispone de un fichero de texto de nombre ``fasta.txt'' con información de una serie de secuencias de ADN, incluyendo su identificador, una descripción y la secuencia propiamente dicha. La primera línea de todas las secuencias, que es la que empieza por el carácter `>', contiene un identificador (primera palabra) seguido de una descripción (resto de palabras). Las líneas a continuación, que pueden ser una o varias y pueden tener hasta 80 caracteres de longitud, contienen la secuencia de ADN. Dicha secuencia está formada únicamente por los caracteres `A', `C', `G' y `T' (que se corresponden con las bases adenina, citosina, guanina y timina, respectivamente). Por ejemplo:

>AF013252 Homo sapiens preprocortistatin (Cort) mRNA, complete cds.
GGCACGAGGCCAAACATTGATTTCAGGGCTGCCAGGAAGGAAGAGCAGCAG
CAGGGTGGGAGAGAAGCTC
AGGCTACAC
>AF086433 Homo sapiens full length insert cDNA clone ZD80A01
GCGTTCCAAAATAAGAGCAAATTCGCTCTAAACACAGGAAAAGACCTGAAG
CTTTAATTAAGGGGTTACA
TCCAACCCCAGAGCGCTTTTGTGGGCACTGATTGCTCCAGCTTCTGCGTCAC
TGCGCGAGGGAAGAGGGA
A
>AB000263 Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCAC
GGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGC
AGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCG
GGCCCCTCATAGGAGAGG
TTTAATTACAGACCTGAA

Implementar en Python:

a) (2 puntos) Una función leer_datos_adn que lea los datos del fichero y los almacene en una matriz (lista de listas) con tantas filas como secuencias de ADN contenga el fichero y con 3 columnas, una para cada uno de los ítems que se describen arriba. La función deberá devolver esta matriz como resultado.

b) (2 puntos) Una función son_complemento_inverso que dadas dos secuencias de ADN determine si son una complemento inverso de la otra. La función deberá devolver un booleano. El complemento inverso de una secuencia de ADN es otra secuencia donde las bases se sustituyen por su complemento y además aparecen en orden inverso. El complemento de una base se define como sigue: el complemento de `C' es `G', el de `G' es `C', el de `A' es `T' y el de `T' es `A'. Por ejemplo, dadas las secuencias `CATGATTTT' y `AAAATCATG', que son una complemento inverso de la otra, la función deberá devolver True.