ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
CCPG1001 - FUNDAMENTOS DE PROGRAMACIÓN
SEGUNDA EVALUACIÓN - II TÉRMINO 2017-2018/ Febrero 14, 2018

Nombre: _____ ***Matrícula:*** _____ ***Paralelo:***____

COMPROMISO DE HONOR: Al firmar este compromiso, reconozco que el presente examen está diseñado para ser resuelto de manera individual, que puedo usar un lápiz o esferográfico; que sólo puedo comunicarme con la persona responsable de la recepción del examen; y, cualquier instrumento de comunicación que hubiere traído, debo apagarlo y depositarlo en la parte anterior del aula, junto con algún otro material que se encuentre acompañándolo. Además no debo usar calculadora alguna, consultar libros, notas, ni apuntes adicionales a los que se entreguen en esta evaluación. Los temas debo desarrollarlos de manera ordenada.
Firmo el presente compromiso, como constancia de haber leído y aceptado la declaración anterior. "Como estudiante de ESPOL me comprometo a combatir la mediocridad y actuar con honestidad, por eso no copio ni dejo copiar".

_____
Firma

TEMA 1 (35 PUNTOS)

Para match.com es importante emparejar personas para este día del Amor y de la Amistad. Para esto cuenta con un diccionario **`personas`** con la siguiente estructura:

```
personas = {
   'P1021' : {
              'Nombre': 'Carlos S.',
              'caracteristicas': {'alegre', 'fumador', 'hacker', 'deportista'},
              'citas': {
                    'fallidas': ['P1902','P2893', 'P2310'],
                    'exitosas': ['P4025', 'P1001'] }
              },
      ...
 'P1001' : {
              'Nombre': 'Andrea V.',
              'caracteristicas': {'farrero', 'programador', 'fabuloso', 'deportista'},
              'citas': {
                    'fallidas': ['P1802'],
                    'exitosas': ['P1021', 'P1002'] }
              },
}
```

Para conocer la compatibilidad entre dos personas se utilizan sus *características* y se calcula el índice de Tanimoto de la siguiente manera:

$$Tanimoto(P_1, P_2) = \frac{|caracteristicasP_1 \cap caracteristicasP_2|}{|caracteristicasP_1 \cup caracteristicasP_2|}$$

Implemente lo siguiente:

1. [10 puntos] La función **hayEmparejamiento(codigoP1, codigoP2, dicPersonas, aceptacion)** que recibe el código de dos personas, el diccionario de personas y el nivel de aceptación (entre 0 y 1). Esta función devolverá una tupla con el valor del índice de Tanimoto y un valor de verdad *True* en el caso de que haya emparejamiento y *False* en caso contrario. Hay emparejamiento cuando el valor del índice de Tanimoto es superior o igual al nivel de aceptación y no han tenido una cita previa (exitosa o fallida).

2. [13 puntos] La función **imprimirResultados(codigoPersona, dicPersonas, aceptacionMinimo, aceptacionMaximo)** recibe el código de una persona, el diccionario de personas y los niveles de aceptación mínimo y máximo. Esta función recorre todo el diccionario de personas y genera un reporte en un archivo. El nombre del archivo es el **codigoPersona** con extensión ".txt".

   El formato de cada línea del archivo es el siguiente:

   codigo$nombre$caracteristica1,caracteristica2,..$indiceTanimoto$textoCompatibilidad

   Donde *textoCompatibilidad* debe ser "aceptar" si el índice de Tanimoto está entre los niveles de aceptación mínimo y máximo, caso contrario debe ser "rechazar".

3. [12 puntos] La función **compatibles(codigoPersona, dicPersonas, aceptacion)** recibe el código de una persona, el diccionario de personas y el nivel de aceptación. Defina **aceptacion** como un parámetro con **valor por defecto de 0.43**. Esta función recorrerá **personas** y devolverá otro diccionario cuyas claves correspondan a los códigos de los candidatos con las que **codigoPersona** tiene emparejamiento, considerando el nivel de aceptación enviado como parámetro. El valor corresponde a un diccionario con el nombre, características, el índice de Tanimoto y la cantidad de citas fallidas. Ejemplo del valor a retornar:

```
retorno = {
   'P1021' : {
              'Nombre': 'Carlos S.',
              'caracteristicas': {'alegre', 'fumador', 'hacker', 'deportista'},
              'indice': 0.75,
              'fallidas': 3
              },
       ...
   'P1001' : {
              'Nombre': 'Andrea V.',
              'caracteristicas': {'farrero', 'programador', 'fabuloso', 'deportista'},
              'indice': 0.98,
              'fallidas': 1
              },
}
```

TEMA 2 (55 PUNTOS)

Para el intercambio comercial entre países se tienen archivos con las transacciones de venta. Las transacciones están almacenadas en un archivo diferente por cada categoría (Flores, Frutas, Maderas, etc.) de producto. Por ejemplo, las ventas de claveles, rosas, tulipanes, girasoles, etc. están almacenadas en el archivo "Flores.txt"; las ventas de cacao, banana, etc. están almacenadas en el archivo "Frutas.txt"

Flores.txt

```
Comprador,Vendedor,Producto,UnidadesVendidas,VentasEn$,Fecha
Estados Unidos,Ecuador,rosas,59284,631432.21,2018-01-10
Holanda,Japon,tulipanes,2384,12434.87,2017-11-22
…
Estados Unidos,Ecuador,girasoles,38284,331432.75,2018-02-01
```

Note que un país puede vender el mismo producto al mismo comprador pero en una fecha diferente.

Desarrolle lo siguiente:

1. [15 puntos] La función **calculaTotales(categoria)** que recibe el nombre de una categoría. La función deberá leer el archivo para esa categoría y retornar un diccionario de totales con la siguiente estructura:

   ```
   totales = {(comprador,vendedor,producto):totalUnidades}
   ```

   Por ejemplo, si en la categoría "Flores" Estados Unidos le compró a Ecuador rosas en 12 fechas diferentes, en el diccionario deberá aparecer un solo item con las sumas totales de esas 12 transacciones.

   ```
   {('Estados Unidos','Ecuador','rosas'):257868}
   ```

2. [15 puntos] La función **consolidado(nomArchivo, categorias)** que recibe una lista de categorías y genera un archivo con nombre **nomArchivo** en el que se listen todos los totales de unidades vendidas acumulados por Comprador, Vendedor, Producto. El archivo tendrá la siguiente estructura:

   ```
   Comprador,Vendedor,Categoria,Producto,TotalUnidadesVendidas
   ```

Para el resto del ejercicio asuma que tiene una función **crearMatriz** que recibe el nombre del archivo consolidado y devuelve una tupla con tres elementos: (1) matriz M cuyas filas representan países vendedores, columnas representan productos ordenados alfabéticamente dentro de cada categoría y las celdas representan ventas totales en unidades, (2) lista con las etiquetas de las filas y (3) lista con las etiquetas de las columnas. Las categorías no están ordenadas alfabéticamente.

3. [25 puntos] La función **ventasCategorias(nomArchivo, dicCat)** que recibe el nombre del archivo consolidado y un diccionario donde las claves son las categorías y los valores son listas con todos los productos ordenados alfabéticamente dentro de cada categoría. La función deberá generar por cada categoría un archivo con el nombre de la categoría y extensión ".txt". Cada archivo debe contener los **5 países que han vendido menos productos para esa categoría**. Cada archivo tendrá la siguiente estructura:

   ```
   País,Total_Ventas
   ```

TEMA 3 (10 PUNTOS)
Considere la siguiente matriz **M** de Numpy para los numerales a continuación.

'h'	'o'	'l'	'a'
'm'	'u'	'n'	'd'
'o'	'E'	'S'	'P'
'O'	'L'	':'	')'

1. Indique la salida por pantalla del siguiente código. Justifique su respuesta.

```
f,c = M.shape
t = ''
for j in range(c):
    t = t + ''.join(M[:,j].tolist())

print(t)
```

2. Indique la salida por pantalla del siguiente código. Justifique su respuesta.

```
print((((M == 'o') | (M == 'O')).sum(axis = 0) > 0).sum())
```

---//---

Cheat Sheet. Funciones y propiedades de referencia en Python.

Librería Numpy para ***arreglos*:**	para ***listas***:	para ***cadenas***:
np.array((numRows,numCols),dtype=) ***arreglos***.shape ***arreglos***.reshape() np.sum(***arreglos***) np.mean(***arreglos***) np.where(**condición**) np.argsort(**arreglo**) np.unique(**arreglo**) ***arreglos***.sum(axis=1)	***listas***.append(...) ***listas***.count(...) ***listas***.index(...) ***listas***.pop() *elemento* in ***listas***	***cadenas***.islower() ***cadenas***.isupper() ***cadenas***.lower() ***cadenas***.upper() ***cadenas***.split(...) ***cadenas***.find(...) ***cadenas***.count(...)