ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
CCPG1001 - FUNDAMENTOS DE PROGRAMACIÓN
SEGUNDA EVALUACIÓN - II TÉRMINO 2016-2017/ Febrero 14, 2017

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. (30 PUNTOS)

Se le ha encargado la tarea del control del tránsito. Para esto, la ciudad será representada siempre por una matriz de 5x5, dividida en cuadrantes y sectores, donde se registrará los valores de las multas generadas. Cada celda de la matriz corresponde a un cuadrante y registrará el total de multas generadas para ese cuadrante. Habrán cinco sectores: Norte, Sur, Centro, Este y Oeste, que agruparán varios cuadrantes, de acuerdo al esquema mostrado a la derecha:

Esta matriz muestra únicamente la distribución de sectores (no debe crear esta matriz)

Norte	Norte	Norte	Norte	Norte
Oeste	Centro	Centro	Centro	Este
Oeste	Centro	Centro	Centro	Este
Oeste	Centro	Centro	Centro	Este
Sur	Sur	Sur	Sur	Sur

Para cumplir con la tarea, deberá implementar lo siguiente:

1. Una función **generaMatriz(listaMultas)** que recibe una lista de tuplas, donde cada tupla es (coordenadaX, coordenadaY, valor_multa), con las coordenadas del cuadrante y el valor de la multa. La función deberá retornar una matriz de Numpy con el valor agregado de las multas generadas para cada cuadrante.

   Por ejemplo, para la lista de multas:
   [(0, 0, 120), (1, 2, 330), (3, 4, 123), (4, 2, 62), (0, 0, 50), (4, 4, 89), (0, 3, 25), (2, 0, 43), (3, 2, 21), (0, 0, 120)]

   Nota: las coordenadas empiezan en 0,0 y se pueden repetir en la lista de tuplas.

   La función retornará:

290	0	0	25	0
0	0	330	0	0
43	0	0	0	0
0	0	21	0	123
0	0	62	0	89

2. Una función **sectorTop(matriz)** que reciba la matriz generada en el tema anterior, calcule el sector con el valor total de multas más alto y retorne una tupla con el nombre del sector (Norte, Sur, Centro, Este, Oeste) y dicho valor.

   Para nuestro ejemplo anterior, la función retornará: (‘Centro’, 351)

TEMA 2. (60 PUNTOS)

Usted escribirá un programa que ayudará a personas alrededor del mundo a decidir cuál es el país al cual quieren ir a vivir cuando se jubilen. Para ello su programa ofrecerá información sobre el costo de vida usando las métricas descritas debajo.

1.) Escriba la función **cargarDatos(nomFile)** que recibe el nombre de un archivo que en cada línea contiene los siguientes campos "ciudad,metrica,valorDeMetrica" (ver ejemplo). La función retorna un diccionario con la estructura descrita a continuación:

```
"datos.txt" contiene:

Cuenca,temperatura,22
Guayaquil,precioCasas,130000
Cuenca,precioCasas,120000
Bogota,precioCasas,100000
Bogota,temperatura,20
Guayaquil,temperatura,29
```

```
cargarDatos("datos.txt") retorna un
diccionario con la siguiente estructura:

{"Guayaquil": {"precioCasas":130000,
 "temperatura":29},
 "Cuenca": {"precioCasas":120000,
 "temperatura":22}, ...}
```

Nota: solo existen dos métricas posibles 'precioCasas' y 'temperatura' y todas las ciudades tienen ambas métricas.

2.) Escriba la función **metricaPais(datos, paises)** que recibe el diccionario **datos** con la estructura del diccionario generado en la función anterior y el diccionario **paises** que tiene como clave el nombre del país y como valor la lista de ciudades para ese país. Esta función calcula el valor promedio de cada métrica por país y retorna un diccionario cuya clave es el país y cuyo valor es otro diccionario con los promedios por métrica. Por ejemplo, para *Guayaquil* y *Cuenca* que pertenecen al mismo país se calcula el promedio de las métricas *precioCasas* y *temperatura* y se lo asigna al país Ecuador:

```
{"Ecuador": {"precioCasas":125000,"temperatura":25.5},
 "Colombia": {"precioCasas":120000,"temperatura":20} }
```

3.) Escriba la función **generaPaises(promedios,metrica,minimo,maximo)** que recibe el diccionario **promedios** con la estructura del diccionario generado en la función anterior, un string denominado metrica que puede ser '*precioCasas'* o '*temperatura'* y un valor **minimo** y un **maximo** para dicha métrica. Esta función busca aquellos países en los cuales el valor de **metrica** esté entre el valor mínimo y máximo dados como argumento y escribe en un archivo el país y el valor de la métrica separados por coma. El nombre del archivo de salida es el mismo nombre de la métrica con la extensión ".csv". Por ejemplo:

generaPaises(proms,"temperatura",23,26) para el ejemplo anterior generaría el archivo

"temperatura.csv" con el siguiente contenido:
Ecuador,temperatura,25.5

TEMA 3 (10 PUNTOS)

Dados los conjuntos:
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8,9}
C = {5, 4, 7, 0, 1}
Y = {len(A),len(B),len(C)}

a. Indique la salida del programa. Justifique su respuesta
X = A | B
Z = X ^ C
E = X - C
print(A.issubset(E))

b. Indique la salida del programa. Justifique su respuesta
Y = A & Y
Z = Y - C
print(Z)

---//---

Cheat Sheet. Funciones y propiedades de referencia en Python.

Librería Numpy para ***arreglos*:**	para ***conjuntos***:	para ***cadenas***:
np.array((numRows,numCols),dtype=) np.argmax(***arreglos***) numpy.sum(***arreglos***) numpy.mean(***arreglos***) ***arreglos***.sum(axis=1)	**|** union **^** diferencia simétrica **-** diferencia **&** intersección	***cadenas***.islower() ***cadenas***.isupper() ***cadenas***.lower() ***cadenas***.upper() ***cadenas***.split(...) ***cadenas***.find(...) ***cadenas***.count(...)