Problemas #2 - *Estructuras de Datos*

Para los problemas a continuación debe cumplir las especificaciones dadas y hacer entraga de cada problema en el formato que se define al final de cada uno con el título de "**Entrega**". Debe hacer entrega de la solución de estos problemas mediante la plataforma del curso, en la tarea correspondiente, con todos los archivos guardados dentro de una carpeta comprimida. Tanto la carpeta como el archivo comprimido deben tener el siguiente formato: `nombre_apellido.[rar|tar.gz|zip]`. Se le pide encarecidamente que cumpla este formato y que los archivos estén **dentro de una carpeta** con el fin de facilitar la revisión.

Para el desarrollo de este set de problemas se espera que se familiarice con el flujo de trabajo de repositorio git local en conexión con un repositorio remoto en Github. Se pide que todas las soluciones a los problemas se suban al repositorio de Github que se indica más adelante, pues desde ahí se realizará la evaluación. Por lo tanto, todo lo que se especifica en la parte de "**Entrega**" debe estar en el repositorio de Github. **No se revisará ningún script subido a la plataforma**.

Problema 1. *Commit inicial*

Primero debe crear un repositorio en Github con el nombre "`setp02`". Luego, cree un repositorio local en una carpeta nueva con el comando `git init`, luego de esto ingrese los siguientes comandos por terminal, estando en la carpeta del repositorio:

- `git config user.name "Su nombre"`
- `git config user.email "tu@ejemplo.com"`
- `git config github.user "tu_usuario"`

Estos comandos agregarán su nombre, e-mail, y usuario de Github a la configuración local del repositorio.

Una vez hecho lo anterior, debe agregar un archivo `hola.py` que imprima sólo "Hola, mundo" y hacer un *commit* con el mensaje "Commit inicial". Finalmente haga *push* a Github.

.....

Problema 2. *Palíndroma*

Cree un programa que tenga una función `es_palindroma(str)` que retorne `True` si el string `str` ingresado es palíndromo o `False` en caso contrario. La función debe utilizar listas para la verificación.

Entrega: script `es_palindroma.py`.

.....

Problema 3. *Funes el memorioso*

En Python los archivos se abren del siguiente modo: `f = open('nombre_archivo', 'r')`. Este comando abre en modo lectura ('r' por *read*) el archivo indicado y guarda su contenido (sin leer) en la variable `f`. El contenido del archivo es **iterable**, es decir, se puede implementar un `for` para recorrer su contenido, que en este caso iría línea por línea. Además, el método `f.read()` retorna un string con **todo** el contenido.

En la plataforma se encuentra el archivo `funes.txt` que contiene el cuento "Funes el memorioso" (1944) de Jorge Luis Borges, y para este problema debe tomar el cuento y utilizar un diccionario para almacenar la cantidad de veces que cada palabra aparece en el cuento. Para esto considere que "Funes", "FUNES" y "funes" cuentan como la misma palabra.

La salida del programa debe ser cada palabra con su respectiva cuenta, ordenadas de mayor a menor.

Entrega: script `funes.py`.

.....

Problema 4. *Coeficiente del demonio 2.0*

Ahora que ya sabe cómo se determinan los coeficientes binomiales, y lo que éstos representan (combinaciones sin repetición donde el orden no importa), se le pide que construya una función `combinatoria2(set, k)` que determine todas las `k`-tuplas que se pueden formar con el conjunto `set`.

La función debe retornar una lista con las tuplas.

Entrega: script `combi2.py`.

.....

Problema 5. *Ca-chi-pún*

En un juego de ca-chi-pún cada jugador elige Piedra (R), Papel (P) o Tijeras (T). Las reglas son: R le gana a T, P le gana a R y T le gana a P. Vamos a codificar un juego de ca-chi-pún como una lista, donde los elementos, a su vez, listas de 2 elementos que codifican el nombre de un jugador y su jugada, como se muestra a continuación:

```
[ [''Armando'', ''P''], [''Dave'', ''T''] ] # Dave gana ya que T > P
```

Realice una función `ganador_cachipun` que tome como argumento una lista de 2 elementos y haga lo siguiente:

- Si el número de jugadores no es igual a 2, levante la excepción "`Número incorrecto de jugadores`". Las excepciones en Python se lanzan así:

```
raise Exception(‘‘Este es el texto de la excepción’’)
```

- Si la jugada de cualquiera de los dos jugadores es algo que no sea "R", "P" o "T" (sin considerar mayúsculas), levante la excepción "`Jugada no válida`".
- En otro caso, returne el nombre y jugada del jugador ganador. Si ambos jugadores tienen la misma jugada, el ganador es el primero.

Entrega: script `cachipun.py`.

Problema sacado del curso "CS169.1x Software as a Service" de EdX (`http://www.edx.org`*).*

.....

Problema 6. *Craps*

"Craps" es un juego de dados donde los jugadores apuestan sobre el resultado de un lanzamiento, o serie de lanzamientos, de un par de dados. Cada lanzamiento pertenece a un conjunto de varios lanzamientos que son usados para resolver las apuestas. Hay sólo 36 posibles lanzamientos de dados, pero sería muy molesto definir el conjunto a mano.

6.1. Primera parte

A continuación se lista una forma de producir el conjunto de todos los lanzamientos con los que se puede definir un juego de "Craps":

- Crea una secuencia llamada `dado` con 13 conjuntos vacíos, para lo cual necesitarás usar una sentencia `for` para evaluar la función constructra de un conjunto 13 veces.
- Escribe dos ciclos `for`, anidados, que iteren a través de lsa 36 combinaciones posibles de dados, creando 2-tuplas. Las 36 2-tuplas comenzarán con `(1, 1)` y terminarán con `(6, 6)`. La suma de los dos elementos es un índice de la secuencia `dado`.
- Agrega cada 2-tupla al conjunto correspondiente dentro de la secuencia `dado`.

El resultado a obtener debería ser lo siguiente:

```
>>> dado[7]
{[(5, 2), (6, 1), (1, 6), (4, 3), (2, 5), (3, 4)]}
```

6.2. Segunda parte

Ahora se pueden definir las reglas como sets a partir de lo anterior:

- En el primer lanzamiento, **pierdes** si lanzas 2, 3 o 12. Esta regla se puede almacenar en el conjunto {`dado[2] | dado[2] | dado[12]`}. El juego termina.
- En el primer lanzamiento, **ganas** si lanzas 7 u 11. El juego termina. Esta regla es el conjunto {`dado[7] | dado[11]`}.
- En el primer lanzamiento, cualquier otro resultado (4, 5, 6, 8, 9 o 10) establece un **punto**. El juego continúa hasta que se obtiene el punto o un 7.
- Una vez que se establece un punto, **ganas** si obtienes el número del punto y **pierdes** si obtienes un 7.

Programe la función `craps` en la que se genere de forma aleatoria el primer lanzamiento y que se defina si el juego termina ganando, perdiendo, o continúa. El programa debe mostrar el lanzamiento y si se gana o se pierde. Si el juego continúa, se debe mostrar la misma salida.

Entrega: script `craps.py`.

.....