ITCR
ELEMENTOS DE COMPUTACIÓN – I SEMESTRE 2022

TAREA 5: PRÁCTICA MATRICES, DICCIONARIOS, LISTAS, TUPLAS, STRINGS

INSTRUCCIONES:

- Desarrollar en Python 3 en un archivo de nombre: tarea5_su_nombre.py.
- Use el material estudiado a la fecha de esta evaluación. Valide las restricciones solo cuando se pida explícitamente.
- Enviar la solución al tecDigital EVALUACIONES / TAREAS. Revise que el trabajo enviado sea el correcto y a este destino.
- Requisitos para revisar la evaluación:
    - Las funciones deben tener: los nombres indicados en cada ejercicio, el número de cada ejercicio y deben estar en la solución ordenados de acuerdo a su número de ejercicio.
    - Buenas prácticas básicas de programación:
        - Las funciones deben tener documentación interna (COMENTARIOS EN EL PROGRAMA FUENTE), al menos: lo que hace, entradas y sus restricciones, salidas. También ponga documentación interna en aquellas partes que ameriten una explicación adicional para un mejor entendimiento del programa fuente.
        - Nombres significativos de variables y funciones.
        - Reutilizar software: usar funciones que han sido desarrolladas previamente en esta misma evaluación.
- Cada ejercicio vale 10 puntos.
- Fecha de entrega: 1 de junio, 11pm.
- Se coordinará día y hora para revisar la evaluación junto con el estudiante quien siendo el autor mostrará el dominio de la solución implementada desde el punto de vista técnico (uso de conceptos de programación y del lenguaje) así como de la funcionalidad (lo que hace la solución). La revisión puede constar de las siguientes actividades:
    - Revisar esta solución particular
    - Revisar conceptos incluidos en la evaluación
    - Aplicar otras actividades con una complejidad igual o menor a la evaluación.

RECOMENDACIÓN. Siga la metodología de solución de problemas: entender el problema, diseñar un algoritmo, codificar y probar programa. En la etapa de diseño del algoritmo haga un esquema para determinar el comportamiento o patrón del algoritmo, luego proceda con su desarrollo.
NO SE VAYA DIRECTAMENTE A PROGRAMAR EN LA COMPUTADORA, PRIMERO PLANIFIQUE LA SOLUCIÓN HACIENDO ESE ESQUEMA Y LUEGO HAGA EL PROGRAMA.

RECOMENDACIÓN. La universidad le brinda diferentes recursos en este proceso de enseñanza-aprendizaje que usted puede aprovechar: consultas presenciales (vía Zoom), tutorías, asistentes, correos.

Por favor compartan conocimientos e ideas. Para ello pueden trabajar en grupos de 3 personas, pero la presentación y evaluación del trabajo es individual. Es decir que cada miembro del grupo debe enviar el trabajo y su nota será de acuerdo a la revisión particular. Si trabaja en grupos escriba en el fuente su nombre como autor del trabajo y los nombres de los compañeros como coautores.

Ejercicio 1

Desarrolle la función **obtiene_columna** que va a recibir dos argumentos:

- una matriz
- un número de columna.

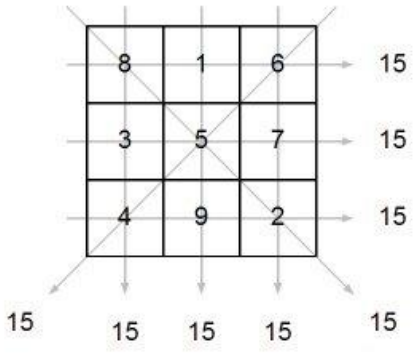
Debe retornar una lista compuesta por los elementos que estén en la columna indicada en el segundo argumento. Si el número de columna (no confundir con el número de índice) no existe en la matriz recibida entonces debe retornar el valor “ERROR: COLUMNA NO EXISTE”

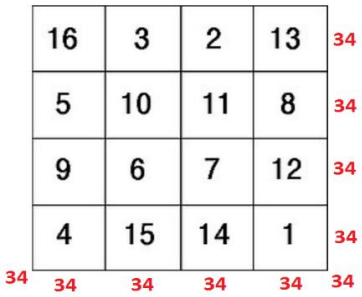
Ejemplos del funcionamiento:

```
>>> obtiene_columna([ [15, 10, 8, 12], [3, 5, 7, 9], [1, 10, 20, 4] ], 1)
[ 15, 3, 1 ]

>>> obtiene_columna([ [8, 10, 8, 12], [3, 5, 7, 9], [1, 10, 20, 4] ], 3)
[ 8, 7, 20 ]

>>> obtiene_columna([ [8, 10, 8, 12], [3, 5, 7, 9], [1, 10, 20, 4] ], 0)
ERROR: COLUMNA NO EXISTE
```

Ejercicio 2

Desarrolle la función **cuadrado_magico** que reciba una matriz cuadrada y retorne True si representa un cuadrado mágico o False de lo contrario. Una matriz representa un cuadrado mágico cuando la suma de los elementos de cada fila y los de cada columna y los de la diagonal principal (desde casilla superior izquierda hasta la inferior derecha) y los de la diagonal secundaria (desde casilla superior derecha hasta la inferior izquierda) dan el mismo valor según muestran los siguientes ejemplos:





8	1	6
3	5	7
4	2	9

Ejemplos del funcionamiento:

```
>>> cuadrado_magico( [ [8, 1, 6], [3, 5, 7], [4, 9, 2] ] )
True

>>> cuadrado magico( [ [16, 3, 2, 13], [5, 10, 11, 8], [9, 6, 7, 12], [4, 15, 14, 1] ])
True

>>> cuadrado_magico( [ [8, 1, 6], [3, 5, 7], [4, 2, 9] ] )
False
```

Ejercicio 3

Haga la función ***números_bonitos*.** Recibe un entero n (>=1) e imprime los primeros n números bonitos. Sea x un número natural, x es bonito si la suma de sus dígitos es igual a la suma de los dígitos de la expresión 3x + 11. Ejemplos de números bonitos:

134 es un número bonito
suma de dígitos de 134: 1 + 3 + 4 = 8
suma de dígitos de fórmula 3 * 134 + 11 = 413: 4 + 1 + 3 = 8
3896 es un número bonito
suma de dígitos de 3896: 3 + 8 + 9 + 6 = 26
suma de dígitos de fórmula 3 * 3986 + 11 = 11969 = 1 + 1 + 9 + 6 + 9 = 26

Validar restricciones: entero y >=1.

Ejemplo del funcionamiento para imprimir los primeros 220 números bonitos:

```
>>> números_bonitos(220)
1 ) 8
2 ) 17
3 ) 35
4 ) 44
5 ) 53
...
219) 3941
220) 3950
```

Ejercicio 4

La sucesión de Fibonacci es la siguiente sucesión infinita de números naturales:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, …

La sucesión empieza con los números 0 y 1, y a partir de estos el siguiente elemento es la suma de los dos números anteriores.

Haga la función **fibonacci** que reciba un entero n (>= 1) y retorne una tupla con los primeros n términos de la sucesión.

Ejemplos del funcionamiento:

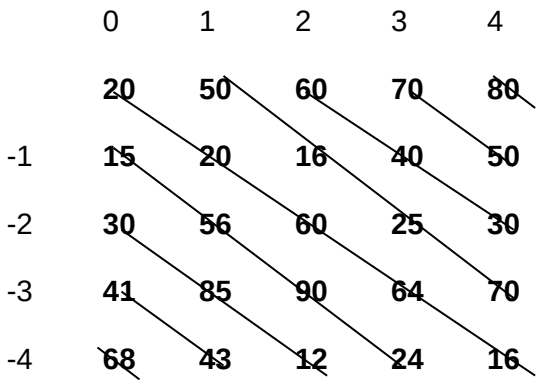
```
>>> fibonacci(5)
(0, 1, 1, 2, 3)

>>> fibonacci(8)
(0, 1, 1, 2, 3, 5, 8, 13)

>>> fibonacci(1)
(0,)
```

Ejercicio 5

Desarrolle la función **suma_diagonal** que reciba dos parámetros:

- Una matriz representada por filas de tamaño m x n (m > 0, n > 0, elementos numéricos)
- Un número de diagonal (entero).

La función debe **retornar** la suma de los elementos que están en el número de diagonal dado en el segundo parámetro. La diagonal principal es la diagonal 0, las diagonales arriba de la principal se numeran de 1 en adelante y las que están debajo de -1 en adelante tal como se muestra a continuación:



Validaciones: la matriz debe ser cuadrada de lo contrario retornar el mensaje “ERROR: NO ES CUADRADA”, la diagonal debe existir de lo contrario retornar el mensaje “ERROR: NO EXISTE LA DIAGONAL”.

Ejemplos del funcionamiento:

```
>>> suma_diagonal([[20, 50, 60, 70, 80], [15, 20, 16, 40, 50], [30, 56, 60, 25, 30], [41, 85, 90, 64, 70], [68, 43, 12, 24, 16]], 2)
130                    # suma de 60, 40, 30

>>> suma_diagonal([[20, 50, 60, 70, 80], [15, 20, 16, 40, 50], [30, 56, 60, 25, 30], [41, 85, 90, 64, 70], [68, 43, 12, 24, 16]], -3)
84                     # suma de 41, 43

>>> suma_diagonal([[20, 50, 60, 70, 80], [15, 20, 16, 40, 50], [30, 56, 60, 25, 30], [41, 85, 90, 64, 70], [68, 43, 12, 24, 16]], 8)
ERROR: NO EXISTE LA DIAGONAL

>>> suma_diagonal([[20, 50, 60], [15, 20, 16]], 1)
ERROR: NO ES CUADRADA
```

Ejercicio 6

Los números primos y semiprimos son altamente utilizados en los campos de la criptografía (codificación de datos) y de la teoría de números. Los números primos son los enteros >= 2 que tienen exactamente dos divisores: el 1 y él mismo, entre ellos: 2, 3, 5, etc. Los números semiprimos (o biprimos) son los enteros >= 2 formados por el producto de dos números primos, entre ellos: 4 (2 x 2), 39 (3 x 13), etc. Haga la función ***semiprimos*** que reciba un intervalo (2 enteros >= 2, el primero menor que el segundo) e imprima estos resultados:

+ cada número semiprimo en el intervalo y los primos que lo forman
+ cantidad total de semiprimos

En caso de no haber semiprimos imprima solamente el mensaje “NO HAY SEMIPRIMOS EN ESTE RANGO”.

Ejemplos del funcionamiento:

```
>>> semiprimos(14, 24)
Semiprimos          Primos que lo conforman
        14                  2 x 7
        15                  3 x 5
        21                  3 x 7
        22                  2 x 11
Cantidad total de semiprimos: 4

>>> semiprimos(17, 20)
Semiprimos          Primos que lo conforman
NO HAY SEMIPRIMOS EN ESTE RANGO
```

Ejercicio 7

Una empresa tiene un grupo de fábricas de producción de celulares. Cada fábrica tiene la producción del mes en un diccionario que tiene la estructura:

- llave: código del celular a producir -tipo string-
- valor: cantidad de celulares a producir -tipo integer-

Ejemplo del diccionario para una fábrica:
{“S10”: 25000, “S15”:30000, “S8”: 10000 }

La empresa requiere que usted desarrolle la función **producción_total**. Va a recibir una tupla donde cada elemento es un diccionario de cada una de sus plantas y retorna otro diccionario donde estén sumarizadas para todas las plantas las cantidades a producir por celular.

Ejemplo del funcionamiento:

```
>>> producción_total( ({“S10”: 25000, “S25”:30000, “S8”: 10000 }, {“S7”: 5000, “S25”:40000 } ) )

{ “S10”:25000, “S25”:70000, “S8”: 10000, “S7”: 5000 }
```

Ejercicio 8

Se tiene el diccionario estudiantes donde cada elemento (llave:valor) tiene la siguiente estructura:

- llave: carnet del estudiante
- valor: lista con el nombre del estudiante y una tupla por cada materia que lleva junto con su nota.

Desarrolle la función **materias** que reciba un diccionario como el anterior y retorne un nuevo diccionario con otra vista de la información donde cada elemento (llave:valor) tiene la siguiente estructura:

- llave: materia
- valor: lista de tuplas, donde cada tupla indica el carnet y nombre del estudiante y la nota que obtuvo en esa materia

Ejemplo del funcionamiento:

```
>>> materias ( { 200058: [“Carlos”, (85, “Ciencias”), (95, “Español”)],
                 200054: [“Roberto”, (90, “Matemáticas”)],
                 200198: [“Martha”, (85, “Biología”), (85, “Matemáticas”)],
                 200110: [“Marvin”, (95, “Español”)],
                 200155: [“Flor”, (90, “Ciencias”)],
                 200167: [“Cecilia”, (85, “Español”)],
                200173: [“Javier”, (90, “Ciencias”)]  } )
{
“Ciencias”: [(200058, “Carlos”, 85), (200155, “Flor”, 90), (200173, “Javier”, 90)],
“Matemáticas”: [ (200054, “Roberto”, 90), (200198, “Martha”, 85) ],
“Biología”: [ (200198, “Martha”, 85) ],
“Español”: [ (200058, “Carlos”, 95), (200110, “Marvin”, 95), (200167, “Cecilia”, 85) ]
}
```

Adicionalmente a este trabajo se recomienda estudiar:

- Materiales vistos a la fecha
- Algoritmos vistos en clase
- Prácticas
- Tareas
- Pruebas cortas
- Otros materiales impresos e Internet

En la prueba corta no entra: archivos, clases, objetos, manejo de excepciones.

Última línea