TEC – Escuela de Computación – Elementos de computación
TAREA 4 PRÁCTICA DE SECUENCIAS PYTHON: LISTAS, TUPLAS, STRINGS

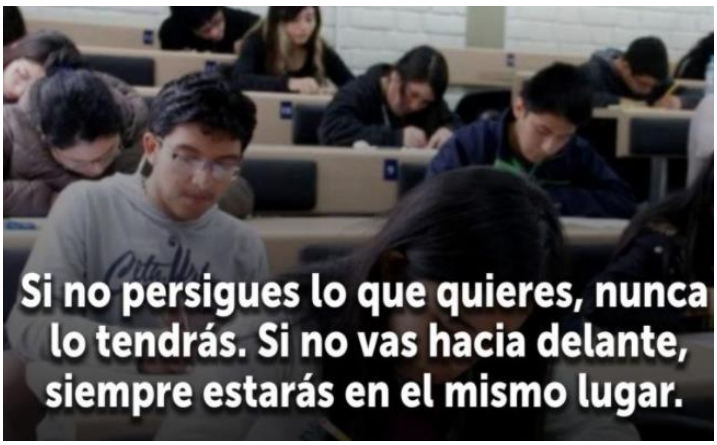
INSTRUCCIONES:

- Desarrollar en Python 3 en un archivo de nombre: tarea4_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: 11 de mayo, 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.



1) Desarrolle la función **obtener_digitos** que reciba una tupla de strings y retorne un string con los dígitos encontrados en la tupla. Los dígitos no deben estar duplicados en el resultado y deben tener el mismo orden que en la tupla. Valide que la entrada sea una tupla y que cada elemento sea un string. Ejemplos del funcionamiento:

>>> obtener_digitos((“am2s48b4”, “hola”, “cod23A3B”))
“2483”

>>> obtener_digitos(“ampmb”)
“”

Haga dos versiones de la función aplicando dos posibles técnicas para recorrer los elementos de una secuencia:
- Usando for (función obtener_digitos_for)
- Usando while (función obtener_digitos_while)

2) Desarrolle la función **elimina_espacios** que reciba un string y retorne 2 valores:
- el string sin los espacios en blanco
- la cantidad de espacios que se eliminaron.

Ejemplo del funcionamiento:

>>> elimina_espacios(“hoy es lunes”)
(“hoyeslunes”, 2)

3) Desarrolle la función **cuenta_palabras** que reciba dos tuplas: una donde cada elemento es una palabra y otra donde cada elemento es una frase. La función debe retornar una lista de tuplas, donde cada tupla va a contener cada palabra de la tupla de palabras de entrada y la cantidad de veces que aparece dicha palabra en la tupla de las frases. Valide que los valores recibidos sean tuplas.

Ejemplo del funcionamiento:

>>> cuenta_palabras((“calor”, “ayer”, “el”, “mañana”), (“ayer hizo bastante calor”, “en el laboratorio hace calor”))

[(“calor”, 2), (“ayer”, 1), (“el”, 1), (“mañana”, 0)]

4) En matemáticas la sucesión de Fibonacci es la sucesión infinita de números naturales:

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

(Número de término: 1 2 3 4 5 6 7 8 9 10 11, …)

Por definición el término número 1 de la sucesión es 0 y el término número 2 es 1. A partir de estos dos términos calculamos el término n-ésimo de la sucesión sumando los dos términos anteriores: término $_{n-1}$ + término $_{n-2}$

Desarrolle la función **fibonacci** que reciba un entero n mayor o igual a 1 y retorne una lista 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(2)
[0, 1]
```

5) Construya la función **tabla_multiplicar**. La función lee tres números enteros (>= 0) e imprime la tabla de multiplicar del primer valor empezando en el segundo valor y terminando en el tercer valor. El segundo valor es <= tercer valor. Ejemplo del funcionamiento:

```
>>> tabla_multiplicar()
Tabla de multiplicar del número: 12
Empieza en el número: 3
Termina en el número: 6
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
```

6) Haga la función **es_palabra_palindroma** que reciba una palabra (string diferente de vacío) y retorne el booleano True si es palíndromo y False en caso contrario.

Una palabra palíndroma es la que tiene el mismo significado si la leemos de izquierda a derecha o de derecha a izquierda. Valide esta restricción: el valor recibido debe ser un string diferente de vacío. Ejemplos del funcionamiento:

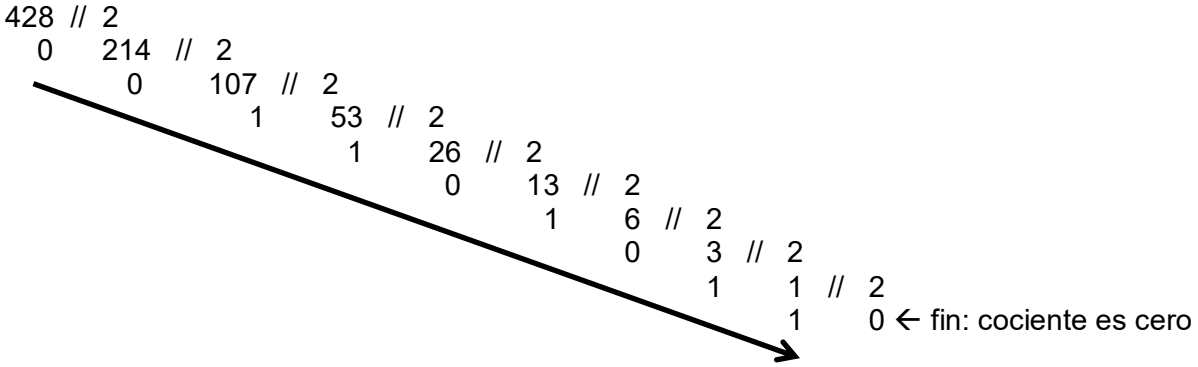
```
>>> es_palabra_palindroma(“reconocer”)
True

>>> es_palabra_palindroma(“sometemos”)
True
```

>>> es_palabra_palindroma(“hola”)
False

>>> es_palabra_palindroma(584)
ERROR: LA ENTRADA DEBE SER UN STRING DIFERENTE DE VACÍO

>>> es_palabra_palindroma(“”)
ERROR: LA ENTRADA DEBE SER UN STRING DIFERENTE DE VACÍO

7) Haga la función **decimal_a_binario** que reciba una lista de números positivos enteros decimales (base 10) y retorne (return) otra lista con sus equivalentes en binario (base 2, la cual usa solo dos dígitos: 0 y 1). Hay varios métodos para hacer esta conversión, vamos a usar el de cocientes sucesivos. Consiste en realizar divisiones enteras sucesivas del número decimal entre 2 (la base a la cual queremos convertir el número decimal) hasta que el cociente sea cero. Los residuos se toman para obtener el número convertido: el residuo de la primera división va a ser el dígito menos significativo del resultado y así sucesivamente. Ejemplo del funcionamiento: convierte el número decimal 428 a su correspondiente número binario: 110101100



Ejemplo del funcionamiento:
>>> decimal_a_binario([428, 4])
[110101100, 100]

Haga dos versiones de la función aplicando dos posibles técnicas para recorrer los elementos de una secuencia:

- Usando for (función decimal_a_binario_for)
- Usando while (función decimal_a_binario_while)

8) Desarrolle la función **lista_cercanos** que reciba una lista de tuplas, cada tupla contiene n números naturales (n >= 2) diferentes entre sí. La función retorna una lista donde cada elemento será una tupla con dos valores asociados por cada elemento en la lista de entrada: el número en la posición n y cuál de los primeros n - 1 números es el más cercano a ese n-ésimo número. En caso de que existan dos números cercanos escoja el número menor.

Ejemplo del funcionamiento para una lista con 4 tuplas y 10 elementos por tupla:

```
>>> lista_cercanos( [ (11, 15, 40, 1, 10, 3, 9, 4, 25, 7 ),
                      (55, 51, 18, 4, 12, 3, 9, 4, 10, 15),
                      (21, 51, 18, 4, 12, 3, 9, 4, 10, 19),
                      (55, 51, 12, 4, 18, 3, 9, 4, 10, 15) ] )

[ (7, 9),       # elemento 1 con resultados para la tupla 1
  (15, 12),     # elemento 2 con resultados para la tupla 2
  (19, 18),     # elemento 3 con resultados para la tupla 3
  (15, 12) ]    # elemento 4 con resultados para la tupla 4
```

9) Usando la instrucción for desarrolle la función **factorial.** El factorial de un número n (denotado como n!) está definido solo para números naturales: es el producto de todos los números enteros desde 1 hasta n:

n! = 1 * 2 * 3 * … * (n – 1) * n

Por definición: 0! = 1

Ejemplos del funcionamiento:

```
>>> factorial(0)   →   1     (por definición es 1)
>>> factorial(1)   →   1     (1*1)
>>> factorial(4)   →   24    (1*2*3*4)
>>> factorial(6)   →   720   (1*2*3*4*5*6)
```

La función recibe un número natural y retorna su factorial.
Validar restricciones (número entero >= 0), sino se cumplen retorna 0.

10) En matemáticas la fórmula de la combinatoria sin repetición es:

$$C_{n,x} = \binom{n}{x} = \frac{n!}{x!\,(n-x)!}$$

donde ***n*** es la cantidad de elementos en un conjunto y ***x*** es el tamaño de cada subconjunto que se puede formar. Cada subconjunto debe ser diferente y el orden de los elementos no interesa.
Ejemplo: en un grupo de 6 estudiantes se quieren hacer subgrupos de 2 estudiantes, ¿cuántos subgrupos diferentes se podrían formar?

$$\frac{6!}{2!\,(6-2)!} = 15$$

Desarrolle la función **combinatoria** que reciba ***n*** y ***x***, retorna el cálculo de la fórmula. Para el ejemplo anterior sería:

```
>>> combinatoria(6, 2)
15
```

Implemente dos soluciones:

a) Para calcular los factoriales: n!, x!, (n – x) ponga la lógica respectiva de cada cálculo.
b) Para calcular los factoriales reutilice software: cada vez que ocupe calcular un factorial llame a la función **factorial**.

11) Desarrolle la función **numeros_bonitos.** Recibe un entero n (>=1) y retorna una lista con 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:

8 es un número bonito
suma de dígitos de 8: 8 = 8
suma de dígitos de la fórmula 3 * 8 + 11 = 35 = 3 + 5 = 8
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

Ejemplo del funcionamiento: crear una lista con los primeros 5 números bonitos:
>>> numeros_bonitos(5)
[8, 17, 35, 44, 53]

12) La lista calificaciones tienen información de los estudiantes en las diferentes materias. Dentro de esta lista cada materia tiene su propia lista con esta estructura:
- nombre de la materia (primer elemento)
- una tupla por cada estudiante que llevó la materia con el carné, nombre del estudiante y la nota en esa materia.

Ejemplo:
[[“Ciencias”, (200058, “Carlos”, 85), (200155, “Flor”, 90), (200173, “Javier”, 90)],
[“Matemáticas”, (200054, “Roberto”, 90), (200198, “Martha”, 85)],
[“Biología”, (200198, “Martha”, 85), (200173, “Javier”, 80)],
[“Español”, (200058, “Carlos”, 95), (200110, “Marvin”, 95), (200167, “Cecilia”, 85)]]

Desarrolle la función **estudiantes_y_materias** que reciba la información anterior y retorne una lista con la información organizada de tal forma que cada estudiante esté representado por otra lista con esta estructura:

- carné del estudiante (primer elemento)
- nombre del estudiante (segundo elemento)
- una tupla por cada materia que llevó el estudiante con la nota y el nombre de la materia

Ejemplo del resultado teniendo como entrada la información anterior:
[[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”), (80, “Biología”)]]

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

Última línea