ITCR - ESCUELA DE COMPUTACIÓN
ELEMENTOS DE COMPUTACIÓN
PRÁCTICA 4: SECUENCIAS

INSTRUCCIONES:

- Desarrolle estas funciones en Python 3 en un solo archivo fuente: **practica4_su_nombre.py**
- Haga la validación de restricciones solo cuando se indique explícitamente.
- Recuerde usar buenas prácticas de programación como documentación interna, nombres significativos, reutilización de código (funciones), eficiencia, etc.
- 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.
- Las funciones deben tener documentación interna (COMENTARIOS EN EL PROGRAMA FUENTE), al menos: lo que hace la función, entradas, salidas.
- Pruebe la ejecución de cada función en el modo comando.
- Enviarla al tecDigital / DOCUMENTOS / EJEMPLOS DE PROGRAMAS.

Sugerencia: siga la metodología de desarrollo, primero entender el problema, desarrollar algoritmo (haga un esquema general de la solución, determine el comportamiento del algoritmo y luego proceda a realizar el detalle de cada parte), codifique y pruebe.

1- Construya la función **indices** que reciba dos valores: un string de 1 carácter y otro string de n caracteres. Retorna una lista con todos los índices donde el primer valor aparece en el segundo valor. Ejemplo del funcionamiento:
>>> indices("x", "axebxx")
[1, 4, 5]

2- Construya la función **diferencia_simetrica** que retorne la diferencia simétrica de dos listas recibidas. La diferencia simétrica de dos listas es una nueva lista con los elementos que pertenecen a alguna de las listas recibidas pero no a ambas. Ejemplo del funcionamiento:
>>> diferencia_simetrica([25, 30, 8, 10, 9], [9, 35, 15, 25, 15])
[30, 8, 10, 35, 15, 15]

3- Construya la función **extraer_numeros** que reciba un string y retorne una lista con cada uno de los números en ese string. Los números que aparecen van a ser solo enteros sin signo y pueden estar representados solos o pegados con otros caracteres.

Ejemplos del funcionamiento:
>>> extraer_numeros("abc25000 articulo 45 codigo xt90z y algunos 400xa65p")
[25000, 45, 90, 400, 65]

>>> extraer_numeros("29 de abril 2018 ")
[29, 2018]

4- Construya la función **blancos** que reciba un string y retorne 2 valores: el string sin los espacios en blanco y la cantidad de espacios que se eliminaron. Valide la restricción: recibe un string. Ejemplo del funcionamiento:
>>> blancos("función para eliminar blancos")
("funciónparaeliminarblancos", 3)

5- Construya la función **contar_palabras** que reciba dos tuplas, una de palabras y otra de frases. Debe retornar una tupla donde cada índice va a tener la cantidad de veces que aparece cada palabra de la tupla de palabras en la tupla de las frases. El índice 1 del resultado corresponde al índice 1 de la tupla de palabras, el índice 2 del resultado corresponde al índice 2 de la tupla de palabras y así sucesivamente. Ejemplo del funcionamiento:

```
>>> contar_palabras( ( “aprender”, “computadoras”, “lenguaje”, “semestre”, “de”),
                     (“este curso es del lenguaje Python”, “Python es un lenguaje de programación
                      de computadoras ”) )
( 0, 1, 2, 0, 2 )
```

La palabra “aprender” se encontró 0 veces, la palabra “computadoras” se encontró 1 vez, etc.

6- En matemáticas un número abundante n cumple la siguiente propiedad: la suma de los divisores positivos de n incluyéndole es mayor al producto de 2 * n. Construya la función es_abundante que reciba un entero > 0 y retorne True si es abundante y False de lo contrario.

Ejemplos del funcionamiento:

```
>>> es_abundante(24)
True
```

24 es abundante porque la suma de sus divisores (1, 2, 3, 4, 5, 8, 12 , 24) es 60, lo cual es mayor al producto de 2 * 24 que es 48.

```
>>> es_abundante(60)
True
```

60 es abundante porque la suma de sus divisores (1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60) es 336, lo cual es mayor al producto de 2 * 60 que es 120.

```
>>> es_abundante(10)
False
```

10 no es abundante porque la suma de sus divisores (1, 2, 5, 10) es 18, lo cual no es mayor al producto de 2 * 20 que es 40.

7- Construya la función **numeros_abundantes** que reciba una tupla de enteros (>= 1) y retorne una lista con los números que son abundantes. Valide que el tipo de datos recibido sea una tupla. Para determinar si un número es abundante use la función es_abundante. Ejemplo del funcionamiento:

```
>>> numeros_abundantes( (24, 60, 10, 38, 18) )
[ 24, 60, 18 ]

>>> numeros_abundantes( [24, 60, 10, 38, 18] )
ERROR: LA ENTRADA DEBE SER UNA TUPLA
```

8- Construya la función **sumar_listas** que reciba dos listas con números y retorne la suma de listas. Para sumar dos listas estas deben tener la misma cantidad de elementos y se suman los índices respectivos (suma del índice i de la primera lista con el índice i de la segunda lista) como muestra el ejemplo de funcionamiento:

```
>>> sumar_listas( [ 10, 1, 30], [ 3, 20, 5])
[ 13, 21, 35]
```

Valide la cantidad de elementos:

```
>>> sumar_listas( [ 10, 1, 30], [ 3, 20 ])
ERROR: PARA SUMAR LISTAS DEBEN TENER LA MISMA CANTIDAD DE ELEMENTOS
```

9- Construya la función **imprimir_palindromos** que reciba un entero n (>= 1) e imprima los primeros n números naturales palíndromos. Valide las restricciones: tipo de datos y valores de los datos, en caso de errores imprima los mensajes respectivos. Puede usar una estructura de programa similar a la usada en el ejercicio anterior, en donde se usó una función adicional (o función auxiliar) para que determine si un número es palíndromo. La impresión que sea como el ejemplo del funcionamiento:

```
>>> imprimir_palindromos(50)
1 .     0
2 .     1
…
49.     393
50      404
```

10- En teoría de números la factorización de enteros o factorización de primos consiste en expresar un número compuesto (número no primo) como un producto de factores primos. Un procedimiento para obtener esta factorización consiste en tomar el número y dividirlo por el menor número primo posible (2 o 3 o 5, etc.), luego el cociente lo dividimos por el mismo primo de ser posible, sino seguimos probando con el siguiente primo, y así sucesivamente hasta que el cociente de 1. Los divisores calculados son los factores primos. Ejemplos:

120	2
60	2
30	2
15	3
5	5
1	

2 x 2 x 2 x 3 x 5

33	3
11	11
1	

3 x 11

12345	3
4115	5
823	823
1	

3 x 5 x 823

911	911
1	

Construya la función **factores_primos** que reciba un entero (> 0) y retorne una lista que represente su factorización como lo indican los ejemplos siguientes. Ejemplos del funcionamiento:

```
>>> factores_primos(120)
[ 2, 2, 2, 3, 5 ]
```

```
>>> factores_primos(33)
[ 3, 11 ]

>>> factores_primos(12345)
[ 3, 5, 823 ]

>>> factores_primos(911)
[ 911 ]
```

ESTUDIE LOS ALGORITMOS VISTOS A LA FECHA DE EJEMPLOS, PRÁCTICAS, TAREAS Y EXÁMEN.

Última línea