RETO-TEST PYTHON: ENUNCIADOS

Ejercicio 1. Función `contar_caracteres`:

Descripción del ejercicio:

- Descripción:
  Crea una función que cuente el número de caracteres en una cadena de texto dada.

Ejercicio 2. Función `calcular_promedio`:

Descripción del ejercicio:

- Descripción:
  Crea una función que calcule el promedio de una lista de números.

Ejercicio 3. Función `encontrar_duplicado`:

Descripción del ejercicio:

- Descripción:
  Crea una función que busque y devuelva el primer elemento duplicado en una lista dada.

Ejercicio 4. Función `enmascarado_datos`:

Descripción del ejercicio:

- Descripción:
  Crea una función que convierta una variable en una cadena de texto y enmascare todos los caracteres con el carácter '#', excepto los últimos cuatro.

Ejercicio 5. Función `es_anagrama`:

Descripción del ejercicio:

- Descripción:
  Crea una función que determine si dos palabras son anagramas, es decir, si están formadas por las mismas letras pero en diferente orden.

Ejercicio 6. Función `buscar_nombre`:

Descripción del ejercicio:

- Descripción:
  Crea una función que solicite al usuario ingresar una lista de nombres y luego solicite un nombre para buscar en esa lista. Si el nombre está en la lista, se imprime un mensaje indicando que fue encontrado, de lo contrario, se lanza una excepción.
- Caso de uso:
  Incorpora los siguientes nombres a la lista y comprueba que la función funciona correctamente: "Jaime", "Silvia" y "Ana".

Ejercicio 7. Función `fibonacci`:

Descripción del ejercicio:

- Descripción:
  Crea una función que calcule el término n de la serie de Fibonacci utilizando recursión.
- Definición de la secuencia de Fibonacci:
  La secuencia de Fibonacci es una serie de números en la que cada número es la suma de los dos números anteriores, comenzando con 0 y 1. La posición 0 es la primera:
- Ejemplo para los primeros 5 términos de la función de Fibonacci: [0, 1, 1, 2, 3]
  - Primer término: 0 (0)

- Segundo término: 1 (1)
- Tercer término: 1 (0 + 1)
- Cuarto término: 2 (1 + 1)
- Quinto término: 3 (1 + 2)

Ejercicio 8. Función `encontrar_puesto_empleado`:

Descripción del ejercicio:

- Descripción:
  Crea una función que tome un nombre completo y una lista de empleados, busque el nombre completo en la lista y devuelve el puesto del empleado si está en la lista, de lo contrario, devuelve un mensaje indicando que la persona no trabaja aquí.
- Caso de uso:

  `empleados` = [{'nombre': "Juan", 'apellido': "García", 'puesto': "Secretario"}, {'nombre': "Mabel", 'apellido': "García", 'puesto': "Product Manager"}, {'nombre': "Isabel", 'apellido': "Martín", 'puesto': "CEO"}]

Ejercicio 9. Función `cubo_numero` usando lambdas:

Descripción del ejercicio:

- Descripción:
  Crea una función que calcule el cubo de un número dado mediante una función
  lambda

Ejercicio 10. Función `resto_division` usando lambdas:

Descripción del ejercicio:

- Descripción:
  Crea una función

`lambda` que calcule el resto de la división entre dos números dados.

Ejercicio 11. Función `numeros_pares` usando lambdas y `filter`:

Descripción del ejercicio:

- Descripción:
  Crea una función
  `lambda` que filtre los números pares de una lista dada.
- Caso de uso: lista_numeros = [24, 56, 2.3, 19, -1, 0]

Ejercicio 12. Función `numeros_suma` usando lambdas y `map`:

Descripción del ejercicio:

- Descripción:
  Crea una función
  `lambda` que sume 3 a cada número de una lista dada.
- Caso de uso: lista_numeros = [24, 56, 2.3, 19, -1, 0]

Ejercicio 13. Función `sumar_listas` usando lambdas:

Descripción del ejercicio:

- Descripción:
  Crea una función
  `lambda` que sume elementos correspondientes de dos listas dadas.
- Caso de uso:
  lista_numeros_1 = [1, 4, 5, 6 , 7 , 7] ;
  lista_numeros_2 = [3, 11, 34, 56]

Ejercicio 14. `No te vayas por las ramas`:

Explicación del ejercicio:

- Descripción:
  Crea la clase
  `Arbol`, define un árbol genérico con un tronco y ramas como atributos. Los métodos disponibles son: `crecer_tronco`, `nueva_rama`, `crecer_ramas`, `quitar_rama` e `info_arbol`. El objetivo es implementar estos métodos para manipular la estructura del árbol.
- Código a seguir:
  1. Inicializar un árbol con un tronco de longitud 1 y una lista vacía de ramas.
  2. Implementar el método `crecer_tronco` para aumentar la longitud del tronco en una unidad.
  3. Implementar el método `nueva_rama` para agregar una nueva rama de longitud 1 a la lista de ramas.
  4. Implementar el método `crecer_ramas` para aumentar en una unidad la longitud de todas las ramas existentes.
  5. Implementar el método `quitar_rama` para eliminar una rama en una posición específica.
  6. Implementar el método `info_arbol` para devolver información sobre la longitud del tronco, el número de ramas y las longitudes de las mismas.
- Caso de uso:
  1. Crear un árbol.
  2. Hacer crecer el tronco del árbol una unidad.
  3. Añadir una nueva rama al árbol.
  4. Hacer crecer todas las ramas del árbol una unidad.
  5. Añadir dos nuevas ramas al árbol.
  6. Retirar la rama situada en la posición 2.
  7. Obtener información sobre el árbol.

Ejercicio 15. Clase `UsuarioBanco`:

Explicación del ejercicio:

- Descripción:
  Crea la clase
  `UsuarioBanco`,representa a un usuario de un banco con su nombre, saldo y si tiene o no cuenta corriente. Proporciona métodos para realizar operaciones como retirar dinero, transferir dinero desde otro usuario y agregar dinero al saldo.
- Código a seguir:
  1. Inicializar un usuario con su nombre, saldo y si tiene o no cuenta corriente mediante `True` y `False`.
  2. Implementar el método `retirar_dinero` para retirar dinero del saldo del usuario. Lanzará un error en caso de no poder hacerse.
  3. Implementar el método `transferir_dinero` para realizar una transferencia desde otro usuario al usuario actual. Lanzará un error en caso de no poder hacerse.
  4. Implementar el método `agregar_dinero` para agregar dinero al saldo del usuario.
- Caso de uso:
  1. Crear dos usuarios: "Alicia" con saldo inicial de 100 y "Bob" con saldo inicial de 50, ambos con cuenta corriente.
  2. Agregar 20 unidades de saldo de "Alicia".
  3. Hacer una transferencia de 80 unidades desde "Bob" a "Alicia".
  4. Retirar 50 unidades de saldo a "Alicia".

Ejercicio 16. Función `procesar_texto`:

Explicación del ejercicio:

- Descripción:
  Crea una función llamada
  `procesar_texto` que procesa un texto según la opción especificada:
  `contar_palabras`, `reemplazar_palabras`, `eliminar_palabra`. Estas opciones son otras funciones que tenemos que definir primero y llamar dentro de la función
  `procesar_texto`.
- Código a seguir:
  1. Crear una función `contar_palabras` para contar el número de veces que aparece cada palabra en el texto. Tiene que devolver un diccionario.
  2. Crear una función `reemplazar_palabras` para remplazar una `palabra_original` del texto por una `palabra_nueva`. Tiene que devolver el texto con el remplazo de palabras.
  3. Crear una función `eliminar_palabra` para eliminar una palabra del texto. Tiene que devolver el texto con la palabra eliminada.
  4. Crear la función `procesar_texto` que tome un texto, una opción(entre "contar", "reemplazar", "eliminar") y un número de argumentos variable según la opción indicada.
- Caso de uso:
  texto = "Este es un ejemplo de texto. Este texto contiene palabras repetidas."

  Dado el texto de ejemplo, llama a la función `procesar_texto`para probar sus funcionalidades:
  - Cuenta el número de veces que aparece cada palabra.
  - Reemplaza la palabra texto por relato.
  - Elimina la palabra ejemplo.