





Python avanzado - ejercicios

 Autoría	 Ana Raquel Martinez Carballo
 Etiquetas	
 Hora de creación	@28 de enero de 2024 23:17

Recursividad:

Tienes a tu disposición un conjunto de discos numerados del 1 al N y tres torres etiquetadas como A, B y C. La torre A contiene inicialmente todos los discos apilados en orden descendente, desde el disco N en la parte inferior hasta el disco 1 en la parte superior.

Tu tarea es implementar una solución **recursiva** para mover todos los discos desde la torre A hasta la torre C, siguiendo las reglas clásicas de las Torres de Hanoi:

1. Puedes mover un disco a una torre adyacente.
2. Solo puedes mover un disco a la cima de otra pila si esa pila está vacía o si el disco superior es más grande que el disco que estás colocando.

Debes definir una función llamada `torres_de_hanoi(n, origen, destino, auxiliar)` que, dado el número total de discos `n` y las torres de origen, destino y auxiliar, imprima los pasos necesarios para lograr el movimiento de todos los discos desde la torre A hasta la torre C.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- N de discos - N de torres - Torres : origen, desitno, auxiliar	Mover disco de la torre A a la torre D Mover disco de la torre A a la torre B ...

Ejercicio 2:

Problema de Gestión de Inventario:

Imagina que trabajas en una empresa de logística y tu tarea es desarrollar un sistema de gestión de inventario. El inventario está representado como una lista

de productos ordenados por sus códigos. Cada producto se describe como un diccionario con las claves `'codigo'` y `'cantidad'`.

Tu objetivo es implementar una función **recursiva** que realice una búsqueda binaria en este inventario y devuelva la cantidad disponible para un producto específico, dado su código.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- Inventario de productos (json,dic,etc) - Código de producto buscado	Cantidad disponible para el producto 307: 80

Ejercicio 3:

Problema de Resolución de Laberinto:

Imagina que eres parte de un equipo de desarrollo de IA que se encarga de crear un sistema para que un robot resuelva laberintos. El laberinto está representado por una matriz, donde ciertos valores indican caminos permitidos (`0`), paredes (`1`), y la salida (`9`). Tu tarea es implementar una función **recursiva** que encuentre la ruta más corta para que el robot salga del laberinto.

Toma en cuenta los siguientes puntos:

1. La matriz representa el laberinto, donde los valores son:
 - `0`: Camino permitido.
 - `1`: Pared, no se puede atravesar.
 - `9`: Salida del laberinto.
2. Debes implementar la función `resolver_labirinto` que utiliza recursividad para encontrar la ruta más corta desde una posición inicial hasta la salida.
3. La función debe devolver una lista de coordenadas que representan la ruta desde la posición inicial hasta la salida.
4. Puedes usar una lista de movimientos posibles: arriba (`(-1, 0)`), abajo (`(1, 0)`), izquierda (`(0, -1)`), derecha (`(0, 1)`).

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
<ul style="list-style-type: none"> - Laberinto (matriz) - Índice de inicio (fila) - Índice de inicio (columna) 	Camino para salir del laberinto: (1,1),(1,2),(),()...

Funciones Lambda:

Ejercicio 4:

Problema de Organización de Datos Empresariales:

Imagina que trabajas en una empresa internacional con equipos distribuidos en diferentes países. Cada equipo tiene una lista de empleados, representados como diccionarios, con información sobre el nombre, la edad y el rendimiento en proyectos recientes.

Tu tarea es organizar una lista consolidada de todos los empleados de la empresa. La organización debe seguir ciertas reglas:

1. Los empleados se deben ordenar por el rendimiento en proyectos recientes de forma descendente.
2. Para aquellos con el mismo rendimiento, se deben ordenar por edad de forma ascendente. Además, deseas agrupar a los empleados por país para un análisis más efectivo. Utiliza funciones **lambda**.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- Registro de empleados (json,dic,etc)	Empleados agrupados

Ejercicio 5:

Problema de Análisis de Datos de Ventas:

Imagina que eres parte de una empresa de comercio electrónico y tienes información detallada sobre las ventas de productos. Cada venta se representa como un diccionario, que incluye el nombre del producto, la fecha de venta, el monto de la venta y la ubicación del comprador. Realiza un análisis avanzado de estas ventas.

1. Filtra las ventas realizadas en el último trimestre del año.
2. Selecciona solo las ventas de productos con un monto superior a \$500.
3. Agrupa las ventas por ubicación del comprador.
4. Calcula el promedio del monto de venta para cada ubicación.
5. Ordena las ubicaciones por el promedio del monto de venta de forma descendente. Utiliza funciones ***lambda***.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- Registro de ventas (json,dic,etc)	Ubicaciones por promedio, ej. : [Chile, Ecuador]

Ejercicio 6:

Problema de Transformación y Filtrado de Nombres:

Imagina que te encuentras desarrollando una herramienta de procesamiento de nombres para una aplicación de gestión de contactos. Tienes una lista de nombres en el formato "Apellido, Nombre", realiza las siguientes tareas:

1. Utiliza la función ***lambda*** para transformar una lista de nombres completos al nuevo formato.
2. Filtra la lista para incluir solo los nombres que contienen al menos dos vocales y tienen una longitud mayor a 10 caracteres.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- Lista de nombres, ej: ["Pérez, Juan", "López, María"]	Nombres filtrados, ej: ['María López', 'José García']

Decoradores:

Ejercicio 7:

Logger con Tiempo de Ejecución

Imagina que estás desarrollando un sistema complejo que incluye múltiples funciones críticas. Para asegurarte de que todo funcione correctamente y para realizar un seguimiento del tiempo de ejecución de estas funciones, decides implementar un **decorador** de registro (logger) con tiempo de ejecución.

El decorador debería realizar las siguientes acciones:

1. Antes de llamar a la función original (puedes incluir cualquier función), debe imprimir un mensaje indicando que la función está a punto de ejecutarse.
2. Después de que la función se haya ejecutado, debe imprimir un mensaje que incluya el tiempo que tardó la función en ejecutarse.
3. Si la función original arroja una excepción, el decorador debe manejarla e imprimir un mensaje adecuado, indicando que se ha producido una excepción.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
-	<ul style="list-style-type: none">- Invocando a la función 'mi_funcion'...- La función 'mi_funcion' ha tardado 0.0012459754943847656 segundos en ejecutarse.- Resultados de la función: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]

Ejercicio 8:

Decorador de Control de Acceso:

Imagina que estás trabajando en el desarrollo de un sistema para una aplicación de gestión de documentos en un entorno empresarial. Deseas implementar un **decorador** llamado `verificar_acceso_entorno` que permita controlar el acceso a funciones según el entorno de ejecución.

El decorador debe realizar las siguientes acciones:

1. Antes de ejecutar la función, verificar si el entorno de ejecución es "producción".

2. Si el entorno es "producción", permitir la ejecución de la función y mostrar un mensaje indicando que el acceso fue permitido en el entorno de producción.
3. Si el entorno no es "producción", evitar la ejecución de la función y mostrar un mensaje indicando que el acceso está restringido a entornos de producción.

Luego, aplica este decorador a dos funciones, `subir_documento` y `eliminar_documento`. Intenta ejecutar estas funciones con diferentes entornos y observa el comportamiento del decorador.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- entorno : 'produccion' / 'desarrollo' - subir_documento("Documento1") -eliminar_documento("Documento1")	- Acceso permitido / rechazado : - Documento subido - Documento eliminado

Ejercicio 9:

Ejercicio: Verificación de Inicio de Sesión con Decorador

Estás desarrollando un sistema de autenticación para una aplicación web y deseas implementar un sistema de inicio de sesión que verifique si las credenciales proporcionadas por el usuario son válidas antes de permitir el acceso a ciertas funciones. Además, deseas que una vez que el usuario haya iniciado sesión correctamente, se le proporcione información personal.

Implementa lo siguiente:

1. Un registro de usuarios que contenga información adicional, como el nombre completo y el correo electrónico.
2. Un decorador llamado `verificar_inicio_sesion` que acepte el nombre de usuario y la contraseña como argumentos. Este decorador verificará si las credenciales proporcionadas son válidas comparándolas con el registro de usuarios. Si las credenciales son válidas, la función decorada se ejecutará y se le pasará como argumento la información personal del usuario.
3. Una función llamada `informacion_usuario` que imprima la información personal del usuario después de que haya iniciado sesión correctamente.

Implementa este sistema de inicio de sesión utilizando **decoradores**.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- informacion_usuario("usuario1", "contrasena123")	- Inicio de sesión exitoso para el usuario usuario1. * Información personal del usuario: * Nombre completo: Juan Pérez - Inicio de sesión fallido. Verifica tu nombre de usuario y contraseña.

Memoización y decoradores:

Ejercicio 10:

Problema de Optimización de Subarreglo:

Imagina que estás trabajando en un sistema de análisis de datos y te han proporcionado una lista de números enteros. Tu tarea es desarrollar una función llamada `max_subarray_sum` que encuentre y devuelva la suma máxima de un subarreglo contiguo en la lista.

Por ejemplo, considera la lista `[1, -2, 3, 10, -4, 7, 2, -5]`. El subarreglo contiguo con la suma máxima es `[3, 10, -4, 7, 2]`, y la suma de esos elementos es `18`. Por lo tanto, la función debería devolver `18` para esta lista.

Implementa la función `max_subarray_sum` y, además, aplica **memoización** para mejorar su eficiencia en el cálculo de subarreglos de suma máxima en listas previamente procesadas.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- arreglo: [1, -2, 3, 10, -4, 7, 2, -5]	- 18

Ejercicio 11:

Ejercicio de Memoización en Costos de Envío

Imagina que estás trabajando en un sistema de gestión de costos de envío para una empresa de logística. El sistema debe calcular el costo de envío para diferentes destinos, distancias y pesos de paquetes. Implementa una función llamada `calcular_costo_envio` que tome como entrada un destino, una distancia en kilómetros y un peso en kilogramos, y devuelva el costo total del envío.

Requerimientos:

1. El costo base de envío es de \$5.0.
2. El costo por kilómetro de distancia es de \$0.1.
3. El costo por kilogramo de peso es de \$0.2.

Implementa la función de manera eficiente utilizando memoización para evitar recalculer el costo para los mismos destinos, distancias y pesos.

A continuacion un ejemplo de una posible entrada y salida de la solucion:

Entrada	Salida
- destino_1 = "Ciudad A" - distancia_1 = 150 - peso_1 = 2.5	- Costo de envío con memoización (Ciudad A): 20.5

Ejercicio 12:

Ejercicio de Memoización en Análisis de Texto

Imagina que estás trabajando en un sistema de análisis de texto que requiere calcular la frecuencia de ocurrencia de palabras en un conjunto de documentos. Implementa una función llamada `calcular_frecuencia_palabras` que tome como entrada un texto y devuelva un diccionario que muestre la frecuencia de cada palabra en el texto.

1. La función debe ser capaz de manejar textos y ser insensible a mayúsculas/minúsculas (por ejemplo, "Hola" y "hola" se consideran la misma palabra).
2. Se deben excluir las palabras comunes (artículos, preposiciones, etc.) que no aportan información relevante al análisis.
3. Utiliza memoización para evitar recalculer la frecuencia de palabras para el mismo texto.

A continuación un ejemplo de una posible entrada y salida de la solución:

Entrada	Salida
- texto 1 - texto 2	- frecuencia de palabras, conteo por palabra.

Tienes a tu disposición un conjunto de discos numerados del 1 al N y tres torres etiquetadas como A, B y C. La torre A contiene inicialmente todos los discos apilados en orden descendente, desde el disco N en la parte inferior hasta el disco 1 en la parte superior.

Tu tarea es implementar una solución **recursiva** para mover todos los discos desde la torre A hasta la torre C, siguiendo las reglas clásicas de las Torres de Hanoi:

1. Puedes mover un disco a una torre adyacente.
2. Solo puedes mover un disco a la cima de otra pila si esa pila está vacía o si el disco superior es más grande que el disco que estás colocando.

Debes definir una función llamada `torres_de_hanoi(n, origen, destino, auxiliar)` que, dado el número total de discos `n` y las torres de origen, destino y auxiliar, imprima los pasos necesarios para lograr el movimiento de todos los discos desde la torre A hasta la torre C.