Ejercicios de Python

Semana 4

14.07.2025 - 20.07.2025

Caso General

Análisis de Interacción en Redes Sociales

Las empresas y creadores de contenido necesitan analizar qué publicaciones generan más interés para entender a su audiencia y mejorar su estrategia de comunicación. Medir "likes", comentarios y la actividad de los usuarios es clave para el crecimiento.

- Yamil Ayma

Nivel Fácil: Filtrar Posts por Autor

Caso

Un community manager quiere revisar rápidamente todas las publicaciones que ha hecho un usuario específico para ver su historial de actividad.

Objetivo

Escribe una función en Python llamada filtrar_posts_por_autor que reciba una lista de posts (lista_posts) y un nombre de autor (nombre_autor). Cada post en la lista es un diccionario. La función debe devolver una nueva lista que contenga únicamente los posts cuyo autor coincida con nombre_autor.

Ejemplo real

Cuando visitas el perfil de un usuario en Instagram o Twitter, la plataforma te muestra un feed que contiene solo las publicaciones de esa persona.

Pruebas de validación

```
def probar_filtrar_posts_por_autor():
    posts = [
        {'autor': 'ana_dev', 'contenido': '¡Aprendiendo Python!', 'likes': 50},
        {'autor': 'beto_data', 'contenido': 'Mi primer análisis de datos.', 'likes': 120},
        {'autor': 'ana_dev', 'contenido': 'Funciones Lambda son geniales.', 'likes': 95}
    ]

    # Prueba 1: Filtrar posts de un autor existente
    posts_de_ana = filtrar_posts_por_autor(posts, 'ana_dev')
    esperado1 = [
        {'autor': 'ana_dev', 'contenido': '¡Aprendiendo Python!', 'likes': 50},
        {'autor': 'ana_dev', 'contenido': 'Funciones Lambda son geniales.', 'likes': 95}
    ]
    print(f"Prueba 1: {posts_de_ana == esperado1}")

    # Prueba 2: Filtrar posts de un autor que no existe
    posts_de_carlos = filtrar_posts_por_autor(posts, 'carlos_design')
    print(f"Prueba 2: {posts_de_carlos == []}")

    # Prueba 3: Filtrar en una lista vacía
    posts_vacios = filtrar_posts_por_autor([], 'ana_dev')
    print(f"Prueba 3: {posts_vacios == []}")

# Descomenta la siguiente línea cuando tengas tu función lista
# probar_filtrar_posts_por_autor()
```

Copie el código

Pegue estas lineas en su archivo Python donde lo desarrollará, debe de validarse exitosamente 👩‍👧

Nivel Medio: Encontrar el Post con Más "Likes"

Caso

Un equipo de marketing necesita identificar cuál de sus publicaciones fue la más popular (recibió más "likes") durante la última campaña para replicar su éxito.

Objetivo

Crea una función llamada encontrar_post_con_mas_likes que reciba una lista_posts. La función debe examinar todos los posts y devolver el diccionario completo del post que tenga el mayor número de 'likes'. Si la lista está vacía, debe devolver None. Si hay un empate, puede devolver cualquiera de los posts con el máximo de "likes".

Continua en la siguiente página

Ejemplo real

Un panel de análisis de redes sociales que muestra la "Publicación con mejor rendimiento" de la semana.

Pruebas de validación

```
def probar_encontrar_post_con_mas_likes():
    posts = [
        {'autor': 'ana_dev', 'contenido': '¡Python es divertido!', 'likes': 50},
        {'autor': 'beto_data', 'contenido': 'Análisis con Pandas.', 'likes': 150},
        {'autor': 'ana_dev', 'contenido': 'Creando una API.', 'likes': 200}
    ]

    # Prueba 1: Encontrar el post con más likes en una lista normal
    top_post = encontrar_post_con_mas_likes(posts)
    esperado1 = {'autor': 'ana_dev', 'contenido': 'Creando una API.', 'likes': 200}
    print(f"Prueba 1: {top_post == esperado1}")

    # Prueba 2: Manejar una lista vacía
    top_post_vacio = encontrar_post_con_mas_likes([])
    print(f"Prueba 2: {top_post_vacio is None}")

    # Prueba 3: Manejar un empate
    posts_empate = [
        {'autor': 'ana_dev', 'contenido': 'Post 1', 'likes': 300},
        {'autor': 'beto_data', 'contenido': 'Post 2', 'likes': 100},
        {'autor': 'ana_dev', 'contenido': 'Post 3', 'likes': 300}
    ]
    top_post_empate = encontrar_post_con_mas_likes(posts_empate)
    es_valido = top_post_empate in [posts_empate[0], posts_empate[2]]
    print(f"Prueba 3 (empate): {es_valido}")


# Descomenta la siguiente línea cuando tengas tu función lista
# probar_encontrar_post_con_mas_likes()
```

Copie el código

Pegue estas lineas en su archivo Python donde lo desarrollará, debe de validarse exitosamente 👷‍♂️👷‍♀️

Nivel Avanzado: Rankear Usuarios por "Engagement"

Caso

Más allá de los "likes", el verdadero valor de una comunidad se mide por el "engagement" total (likes + comentarios). Una empresa quiere identificar a sus usuarios más valiosos, aquellos que no solo publican contenido popular, sino que también generan mucha conversación.

Objetivo

Escribe una función llamada rankear_usuarios_por_engagement que reciba una lista_posts. Cada post es un diccionario que ahora también puede tener una clave 'comentarios', que es una lista de strings.

- Calcula un "puntaje de engagement" para cada autor. El puntaje se define como: total de likes recibidos en todos sus posts + (total de comentarios recibidos en todos sus posts * 2). (Los comentarios valen el doble que los likes).
- La función debe devolver una lista de tuplas, donde cada tupla contiene ('nombre_autor', puntaje).
- La lista debe estar ordenada de mayor a menor según el puntaje de engagement.

Continua en la siguiente página

Ejemplo real

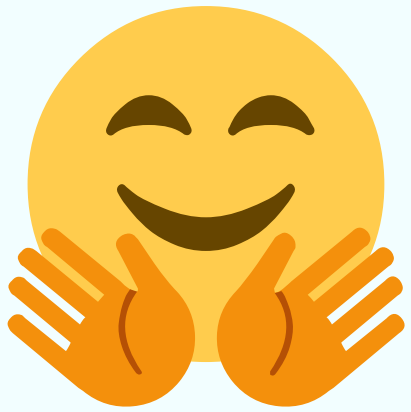
Un sistema de gamificación en un foro que otorga puntos y muestra un "ranking de los mejores contribuidores" para fomentar la participación.

Pruebas de validación

```
def probar_rankear_usuarios_por_engagement():
    posts = [
        {'autor': 'ana_dev', 'contenido': 'Post 1', 'likes': 100, 'comentarios': ['genial', 'útil']},
        {'autor': 'beto_data', 'contenido': 'Post 2', 'likes': 50, 'comentarios': ['interesante']},
        {'autor': 'ana_dev', 'contenido': 'Post 3', 'likes': 150, 'comentarios': ['gracias', 'me
sirvió', 'excelente']},
        {'autor': 'carla_ux', 'contenido': 'Post 4', 'likes': 250, 'comentarios': []}
    ]
    # Puntuaciones:
    # ana_dev: (100 + 150) likes + (2 + 3) comentarios * 2 = 250 + 5*2 = 260
    # beto_data: 50 likes + 1 comentario * 2 = 50 + 2 = 52
    # carla_ux: 250 likes + 0 comentarios * 2 = 250

    ranking = rankear_usuarios_por_engagement(posts)
    esperado = [
        ('ana_dev', 260),
        ('carla_ux', 250),
        ('beto_data', 52)
    ]
    print(f"Prueba 1: {ranking == esperado}")

    # Prueba 2: Lista vacía
    ranking_vacio = rankear_usuarios_por_engagement([])
    print(f"Prueba 2: {ranking_vacio == []}")

# Descomenta la siguiente línea cuando tengas tu función lista
# probar_rankear_usuarios_por_engagement()
```

Copie el código

Pegue estas lineas en su archivo Python donde lo desarrollará, debe de validarse exitosamente 👨‍💻👩‍💻



Éxitos en los ejercicios

No te preocupes si no funciona bien. Si todo estuviera correcto, serías despedido de tu trabajo.
- Ley de ingeniería de software de Mosher

Aprende mucho