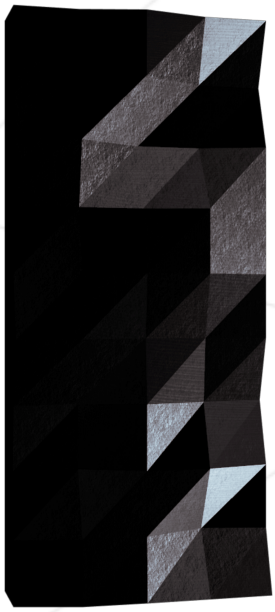

Data Quest

Dominando las colecciones de Python para Ingeniería de Datos

Resumen: ¡Emprende un viaje por el reino digital de la ingeniería de datos! Domina las potentes estructuras de datos de Python mientras creas sistemas de analítica de juegos, procesas estadísticas de juego y construyes pipelines de datos en la Dimensión Píxel.

Versión: 1.0

Índice general

Capítulo I

Prólogo

¡Te damos la bienvenida al reino digital de la ingeniería de datos!

Tu recorrido por los fundamentos de Python te ha preparado bien. Has dominado la sintaxis básica que impulsa los huertos digitales, has construido jerarquías de clases robustas que modelan sistemas del mundo real y has aprendido a manejar lo inesperado con una gestión de excepciones sólida. Ahora, al adentrarte en la **Dimensión Píxel**, prepárate para abordar el corazón de la ingeniería de datos: **colecciones y estructuras de datos**.

Imagina esto: en 1980, todo el juego de Pac-Man, cada punto, posición de los fantasmas y puntuación cabía en solo 16KB de RAM. ¡Quien programaba tenía que dominar la hechicería de la eficiencia! Descubrieron que organizar los datos no solo iba de ahorrar memoria, sino de desbloquear magia jugable. Avance rápido hasta hoy: Fortnite procesa a más de 10 millones de personas jugando de forma concurrente, cada una generando miles de puntos de datos por segundo. ¡Mismos principios, un patio de juegos mucho mayor!

Los tipos de colecciones de Python son como el inventario de tus juegos: las **listas** son tu mochila de confianza (ordenadas, expandibles), los **tuples** son tu armadura equipada (inmutables, fiables), los **sets** son tu colección de logros (únicos, sin duplicados) y los **diccionarios** son tu libro de hechizos (búsquedas instantáneas por nombre). ¡Cada uno tiene su superpoder!

Pero aquí es donde se vuelve épico: los **generadores** y las **comprensiones** son los movimientos combinados definitivos de Python. Imagina procesar un millón de registros en un juego sin despeinarte: ¡esa es la magia de los generadores! O transformar datos complejos con una línea elegante: ¡ese es el poder de las comprensiones!

En esta misión, construirás el "PixelMetrics 3000", una plataforma de analítica de juegos que haría sentir envidia incluso a quienes desarrollaron Minecraft. Cada ejercicio desbloquea un nuevo superpoder de datos y, al final, blandirás las colecciones de Python como una auténtica leyenda de la ingeniería de datos.

Capítulo II

Instrucciones sobre la IA

Contexto

Durante tu proceso de aprendizaje, la IA puede ayudarte con muchas tareas diferentes. Tómate el tiempo necesario para explorar las diversas capacidades de las herramientas de IA y cómo pueden apoyarte con tu trabajo. Sin embargo, siempre debes abordarlas con precaución y evaluar de forma crítica los resultados. Ya sea código, documentación, ideas o explicaciones técnicas, nunca podrás saber con total certeza si tu pregunta está bien formulada o si el contenido generado es el adecuado. Las personas que te rodean son tu recurso más valioso para ayudarte a evitar errores y puntos ciegos.

Mensaje principal:

- Utiliza la IA para reducir las tareas repetitivas o tediosas.
- Desarrolla habilidades de prompting, ya sea para programacion o para otros temas, que beneficiarán tu futura carrera.
- Aprende cómo funcionan los sistemas de IA para anticipar de forma eficiente y evitar los riesgos comunes, sesgos y problemas éticos.
- Sigue trabajando con tus compañeros para desarrollar tanto habilidades técnicas como habilidades transversales.
- Utiliza únicamente contenido generado por IA que entiendas completamente y del cual puedas responsabilizarte.

Reglas para estudiantes:

- Debes tomarte el tiempo necesario para explorar las herramientas de IA y comprender cómo funcionan, para poder utilizarlas de manera ética y reducir los sesgos potenciales.
- Debes reflexionar sobre tu problema antes de dar instrucciones a la IA. Esto te ayuda a escribir preguntas, instrucciones o conjuntos de datos más claros, detalladas y relevantes utilizando un vocabulario preciso.

- Debes desarrollar el hábito de revisar, cuestionar y probar sistemáticamente cualquier contenido generado por la IA.
- Debes buscar siempre la revisión de otras personas, no te limites a confiar en tu propia validación.

Resultados de esta etapa:

- Desarrollar habilidades de prompting tanto generales como de ámbito especifico.
- Aumentar tu productividad con un uso eficaz de las herramientas de IA.
- Seguir fortaleciendo el pensamiento computacional, la resolución de problemas, la adaptabilidad y la colaboración.

Comentarios y ejemplos:

- Ten en cuenta que la IA puede no tener la respuesta correcta porque esa respuesta no esté ni siquiera en Internet. Además, si te da soluciones incorrectas, intenta no insistir y busca ayuda entre las personas que te rodean. Vas a ahorrarte tiempo y vas a sumar en compresión.
- Vas a enfretarte con frecuencia a situaciones (como exámenes o evaluaciones) donde debes demostrar una comprensión real. Prepárate, sigue construyendo tanto tus habilidades técnicas como transversales.
- Explicar tu razonamiento y debatir con otras personas suele revelar lagunas en tu comprensión de un concepto. Prioriza el aprendizaje entre pares.
- Lo normal es que la herramienta de IA que utilices no conozca tu contexto específico (a menos que se lo indiques), así que te dará respuestas genéricas. Si buscas información más adecuada y más precisa en relación a tu entorno cercano, confía en el resto de estudiantes.
- Donde la IA tiende a generar la respuesta más probable, el resto de estudiantes puede proporcionar perspectivas alternativas y matices valiosos. Confía en la comunidad de 42 como un punto de control de calidad.

✓ Buenas prácticas:

> Le pregunto a la IA: "¿Cómo pruebo una función de ordenación?"Me da algunas ideas. Las pruebo y reviso los resultados con otra persona. Refinamos el enfoque de manera conjunta.

✗ Bad practice:

> Le pido a la IA que escriba una función completa, la copio y la pego en mi proyecto. Durante la evaluación entre pares, no puedo explicar qué hace ni por qué. Pierdo credibilidad. Suspendo mi proyecto.

✓ Good practice:

> Utilizo la IA para ayudarme a diseñar un parser. Luego, reviso la lógica con otra persona. Encontramos dos errores y lo reescribimos juntos: mejor, más limpio y comprendiendo al 100

✗ Bad practice:

> Dejo que Copilot genere mi código para una parte clave de mi proyecto. Compila, pero no puedo explicar cómo maneja los pipes. Durante la evaluación, no puedo justificarlo y suspendo mi proyecto.

Capítulo III

Introducción

¡Te damos la bienvenida a Data Quest: La Dimensión Píxel!

Has conquistado los básicos de Python, has dominado las clases y has domado las excepciones. Ahora llega la parte divertida: adentrarte en la magia de los datos. Estás a punto de construir el "PixelMetrics 3000", la plataforma de analítica de juegos más épica de este lado del universo digital.

Piénsalo como tu viaje Pokémon, pero en lugar de capturar criaturas, vas recopilando superpoderes de datos:

- **Nivel 0**: Misión de Comandos - Domina la comunicación por línea de comandos
- **Nivel 1**: Procesador de Puntuaciones - Domina las listas analizando puntuaciones de las personas que juegan
- **Nivel 2**: Rastreador de Posición - Usa tuples para navegar los mundos del juego
- **Nivel 3**: Caza de Logros - Aprovecha sets para conseguir logros únicos
- **Nivel 4**: Control de Inventario - Construye sistemas complejos con diccionarios
- **Nivel 5**: Asistente de Transmisión - Procesa datos infinitos con generadores
- **Nivel 6**: Alquimista de Datos - Transforma todo con comprensiones

Cada ejercicio es como desbloquear una nueva habilidad en tu RPG favorito. Empieza simple, sube de nivel gradualmente y, al final, serás Neo de las estructuras de datos en Python.

IMPORTANTE: Este proyecto se centra en el dominio de las estructuras de datos. Tus programas deben demostrar la aplicación de esta ingeniería de datos en situaciones prácticas y, al mismo tiempo, deben enseñar las características únicas de cada tipo de colección.

Capítulo IV

Instrucciones Generales

IV.1. Reglas Generales

- El proyecto debe estar escrito en **Python 3.10+**.
- Tu código debe respetar los estándares de linter **flake8**.
- Tus funciones deben gestionar de manera robusta las excepciones, para evitar caídas.
- Para este proyecto, solo se permite la importación de **sys** para el procesamiento de las líneas de comandos.
- No se permiten operaciones de entrada/salida de archivos: todos los datos deben procesarse en memoria o mediante argumentos por línea de comandos.
- Céntrate en demostrar con claridad los escenarios de uso de las colecciones.
- Muestra tanto operaciones básicas como técnicas avanzadas para cada estructura de datos.

IV.2. Pautas Adicionales

- Crea programas de prueba para verificar la funcionalidad del proyecto (no se entregan ni califican).
- Sube tu trabajo al repositorio Git asignado.
- Solo se calificará el contenido de ese repositorio.

IV.3. Recursos de Prueba

Para ayudarte a probar tus implementaciones y generar datos de muestra, se han proporcionado herramientas útiles en un archivo de pruebas:

- **Archivo de pruebas**: Extrae `data_quest_tools.tar.gz` (proporcionado junto con el proyecto), para acceder a los útiles de prueba

- **Generador de datos principal**: `data_generator.py` - Genera datos listos para pasar por línea de comandos para todos los ejercicios (0-6)
- **Asistente del ejercicio 0**: `exercise_0_help.py` - Descubre `sys.argv` y los argumentos por línea de comandos
- **Asistente del ejercicio 1**: `exercise_1_helper.py` - Analizador de puntuaciones sólido con patrones de datos realistas
- **Asistente avanzado**: `advanced_data_helper.py` - Escenarios de datos complejos y pruebas de rendimiento

Extrae el archivo de herramientas con: `tar -xzf data_quest_tools.tar.gz`

Ejemplos de inicio rápido:

```
# Generate test commands for all exercises
python3 data_generator.py

# Get specific exercise help
python3 exercise_0_help.py
python3 exercise_1_helper.py

# Generate command-line ready data
python3 data_generator.py 1 --count 10 --format argv
```

Estas herramientas generan datos listos para pasar por línea de comandos que puedes copiar y pegar directamente en tu terminal. No se necesitan archivos: todo funciona con los conceptos que ya conoces.

Las herramientas de prueba revelan plantillas refinadas de Python y técnicas de generación de datos. Están diseñadas para usar únicamente conceptos adecuados para tu nivel de aprendizaje actual: no se requiere entrada/salida avanzada de archivos ni procesamiento de JSON.

Capítulo V

Ejercicio 0: Misión de Comandos

	Ejercicio: 0
ft_command_quest	
Directorio de entrega: *ex*0/	
Archivos a entregar: `ft_command_quest.py`	
Funciones autorizadas: `import sys, sys.argv, len(), print()`	

¡Te damos la bienvenida a la Aventura de Datos! Toda misión épica empieza entendiendo tus herramientas. En el reino digital, los programas necesitan recibir instrucciones del mundo exterior. Tu primera misión: ¡descubrir cómo pueden recibir los programas mensajes de las personas usuarias!

Tu misión: Construye un intérprete de comandos simple que demuestre que dominas el arte de recibir datos externos. Piensa en ello como si tuvieras que aprender a leer qué misiones ha elegido quien juega para ver cómo avanza.

¿Dónde está la magia?:

- Descubrir cómo los programas pueden recibir información desde la línea de comandos.
- Aprender a procesar distintos tipos de datos de entrada.
- Gestionar los casos en los que no se proporciona información.
- Mostrar la información de forma clara para quien la recibe.

Poción potenciadora: Cada programa es como un personaje en un RPG: necesita saber qué quiere la persona que juega que haga. La línea de comandos es cómo comunican estos deseos.

```
$> python3 ft_command_quest.py
=== Command Quest ===
No arguments provided!
Program name: ft_command_quest.py
Total arguments: 1

$> python3 ft_command_quest.py hello world 42
=== Command Quest ===
Program name: ft\_command\_quest.py
Arguments received: 3
Argument 1: hello
Argument 2: world
Argument 3: 42
Total arguments: 4

$> python3 ft_command_quest.py "Data Quest"
=== Command Quest ===
Program name: ft_command_quest.py
Arguments received: 1
Argument 1: Data Quest
Total arguments: 2
```

¿Cómo sabe tu programa qué quiere quien lo usa? ¿Cuál es la diferencia entre el nombre del programa y los argumentos?

Capítulo VI

Ejercicio 1: Procesador de Puntuaciones

	Ejercicio: 1
ft_score_analytics	
Directorio de entrega: *ex*1/	
Archivos a entregar: `ft_score_analytics.py`	
Funciones autorizadas: `sys.argv, len(), sum(), max(), min(), int(), print(), try/except`	

Informe de misión: Ahora que dominas la comunicación por comandos, llega tu primera misión real de datos. PixelMetrics 3000 necesita un módulo Score Cruncher. Es como la *tabla de clasificación* de tu juego favorito, pero tú construyes el motor que la impulsa.

Tu misión (si decides aceptarla):

- Aceptar puntuaciones desde la línea de comandos (como *cheat codes*, pero legales).
- Usar **listas** para almacenar y organizar las puntuaciones.
- Calcular métricas básicas que harían feliz a cualquiera que desarrolle juegos.
- Gestionar adecuadamente los casos "ups, escribí 'banana' en lugar de '1000'".
- Ajustar la salida de datos con un formato agradable que cause buena impresión.

Poción potenciadora: Las listas son como tu inventario en un RPG: puedes añadir elementos, contarlos, encontrar el mejor y organizarlos como quieras.

```
python3 ft_score_analytics.py 1500 2300 1800 2100 1950
```

```
$> python3 ft_score_analytics.py 1500 2300 1800 2100 1950
=== Player Score Analytics ===
Scores processed: [1500, 2300, 1800, 2100, 1950]
Total players: 5
Total score: 9650
Average score: 1930.0
High score: 2300
Low score: 1500
Score range: 800

$> python3 ft_score_analytics.py
=== Player Score Analytics ===
No scores provided. Usage: python3 ft_score_analytics.py <score1> <score2> ...
```

¿Cómo ayudan las **listas** a procesar datos secuenciales? ¿Qué las hace perfectas para el procesamiento de datos desde la línea de comandos?

Capítulo VII

Ejercicio 2: Rastreador de Posición

	Ejercicio: 2
ft_coordinate_system	
Directorio de entrega: *ex2/*	
Archivos a entregar: `ft_coordinate_system.py`	
Funciones autorizadas: `import sys, sys.argv, import math, tuple(), int(), float(), print(), split(), try/except, math.sqrt()`	

¡Subiste de nivel! ¡Es hora de dominar las coordenadas 3D! ¿Recuerdas jugar a videojuegos de mundo 3D donde te teletransportas a ubicaciones específicas? ¿O cuando necesitas hallar la distancia entre dos puntos en un mapa 3D? ¡Eso es exactamente lo que vamos a construir!

Tu misión: Construir un sistema de coordenadas 3D usando **tuples**. Piensa en los tuples como coordenadas GPS que no pueden cambiarse por accidente: perfectas para definir posiciones en un juego.

Qué hace que esto sea divertido:

- Crear posiciones 3D, como por ejemplo puntos de aparición del juego: `(x, y, z)`
- Calcular distancias usando la fórmula euclidiana 3D: `sqrt((x2-x1)² + (y2-y1)² + (z2-z1)²)`
- Analizar cadenas de coordenadas (¡como comandos de teletransporte!)
- Enseñar la magia del desempaquetado de tuples (¡es como abrir un regalo!)

Poción Potenciadora: Los tuples son como coordenadas escritas en piedra: una vez creadas, no cambian. Perfectas para posiciones 3D, colores o cualquier dato que deba mantenerse constante.

Fórmula de distancia explicada: Para calcular la distancia entre dos puntos 3D, se usa la fórmula de distancia euclidiana. Para los puntos `(x1, y1, z1)` y `(x2, y2, z2)`, la distancia es `math.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)`. Es la extensión 3D del Teorema de Pitágoras.

¡Recuerda gestionar adecuadamente el clásico error "escribí 'abc' en lugar de '123'"!

```
$> python3 ft_coordinate_system.py
=== Game Coordinate System ===

Position created: (10, 20, 5)
Distance between (0, 0, 0) and (10, 20, 5): 22.91

Parsing coordinates: "3,4,0"
Parsed position: (3, 4, 0)
Distance between (0, 0, 0) and (3, 4, 0): 5.0

Parsing invalid coordinates: "abc,def,ghi"
Error parsing coordinates: invalid literal for int() with base 10: 'abc'
Error details - Type: ValueError, Args: ("invalid literal for int() with base 10: 'abc'",)

Unpacking demonstration:
Player at x=3, y=4, z=0
Coordinates: X=3, Y=4, Z=0
```

¿Por qué los **tuples** son perfectos para coordenadas de datos 3D? ¿Cómo hace el **desempaquetado** tu código más legible y potente?

Capítulo VIII

Ejercicio 3: Caza de Logros

	Ejercicio: 3
ft_achievement_tracker	
Directorio de entrega: *ex*3/	
Archivos a entregar: `ft_achievement_tracker.py`	
Funciones autorizadas: `set()`, `len()`, `print()`, `union()`, `intersection()`, `difference()`	

¡Logro desbloqueado! ¡Hora de construir el sistema de logros más genial! ¿Conoces la satisfacción de conseguir ese logro tan raro? Ahora te toca a construir el sistema que los registra todos.

Tu misión: Crear un *Achievement Hunter* usando **sets**, la herramienta perfecta para manejar colecciones únicas. En el salón de la fama no se permiten duplicados.

Qué hace que esto sea épico:

- Registrar logros únicos (¡no contar "Primera baja" dos veces!).
- Encontrar logros compartidos para varias personas (el "terreno común").
- Detectar los logros ultrarraros (material para presumir).
- Ver quién tiene logros pendientes (¡hay que conseguirlos todos!).
- Formar comunidades entre las personas que juegan basadas en sus logros compartidos.

Poción Potenciadora: Los sets son como tu vitrina de trofeos: cada logro aparece exactamente una vez y puedes comprobar al instante si lo tienes o comparar colecciones con tus amistades.

```
$> python3 ft_achievement_tracker.py
=== Achievement Tracker System ===

Player alice achievements: {'first_kill', 'level_10', 'treasure_hunter', 'speed_demon'}
Player bob achievements: {'first_kill', 'level_10', 'boss_slayer', 'collector'}
Player charlie achievements: {'level_10', 'treasure_hunter', 'boss_slayer', 'speed_demon', '
    perfectionist'}

=== Achievement Analytics ===
All unique achievements: {'boss_slayer', 'collector', 'first_kill', 'level_10', 'perfectionist', '
    speed_demon', 'treasure_hunter'}
Total unique achievements: 7

Common to all players: {'level_10'}
Rare achievements (1 player): {'collector', 'perfectionist'}

Alice vs Bob common: {'first_kill', 'level_10'}
Alice unique: {'speed_demon', 'treasure_hunter'}
Bob unique: {'boss_slayer', 'collector'}
```

¿Cómo hacen los **sets** que la deduplicación sea trivial? ¿Por qué las operaciones de set son perfectas para analítica?

Capítulo IX

Ejercicio 4: Control de Inventario

	Ejercicio: 4
ft_inventory_system	
Directorio de entrega: *ex4/*	
Archivos a entregar: `ft_inventory_system.py`	
Funciones autorizadas: `dict()`, `len()`, `print()`, `keys()`, `values()`, `items()`, `get()`, `update()`	

¡Botín a la vista! ¿Recuerdas organizar el inventario en los RPG? Clasificar pociones, contar oro, comprobar si tenías esa espada legendaria. Es momento de construir el sistema de inventario definitivo.

Tu misión: Crear un *Inventory Master* usando **diccionarios**, tu sistema de almacenamiento mágico donde puedes encontrar al instante cualquier elemento por su nombre.

Qué hace que esto sea excelente:

- Gestionar los inventarios de las personas (como tu cofre del tesoro personal).
- Registrar detalles de los objetos: cantidades, tipos, valores (¿merece la pena guardarlo?).
- Calcular el valor total del inventario (¿qué riqueza tienes?).
- Organizar elementos por categorías (armas, pociones, armaduras, etc.).
- Generar informes de inventario llamativos (¡presume de tu colección!).

Poción potenciadora: Los diccionarios son como una mochila superorganizada en la que puedes tomar cualquier elemento al instante con solo decir su nombre. No más rebuscar entre todo para encontrar esa poción de salud.

```
$> python3 ft_inventory_system.py
=== Player Inventory System ===

=== Alice's Inventory ===
sword (weapon, rare): 1x @ 500 gold each = 500 gold
potion (consumable, common): 5x @ 50 gold each = 250 gold
shield (armor, uncommon): 1x @ 200 gold each = 200 gold

Inventory value: 950 gold
Item count: 7 items
Categories: weapon(1), consumable(5), armor(1)

=== Transaction: Alice gives Bob 2 potions ===
Transaction successful!

=== Updated Inventories ===
Alice potions: 3
Bob potions: 2

=== Inventory Analytics ===
Most valuable player: Alice (850 gold)
Most items: Alice (5 items)
Rarest items: sword, magic_ring
```

¿Por qué los **diccionarios** son esenciales para los datos de juego?
¿Cómo modelan las relaciones complejas los diccionarios anidados?

Capítulo X

Ejercicio 5: Asistente de Transmisión

	Ejercicio: 5
ft_data_stream	
Directorio de entrega: *ex*5/	
Archivos a entregar: `ft_data_stream.py`	
Funciones autorizadas: `yield, next(), iter(), range(), len(), print(), for loops`	

¡Hora de la magia! ¿Alguna vez te has preguntado cómo los juegos gestionan millones de eventos sin colapsar? Te damos la bienvenida al mundo de los **generadores**, el superpoder de ahorro de memoria de Python.

Tu misión: Construir un *Stream Wizard* que procese datos con maestría. Piensa en los generadores como hechizos que crean datos bajo demanda en lugar de almacenarlo todo de una vez.

¿Dónde está la magia?:

- Crear flujos de datos que fluyen como un río (¡no un lago!).
- Procesar eventos uno a uno usando **bucles for-in** (como leer un libro página a página).
- Filtrar eventos interesantes (¡solo lo bueno!).
- Llevar estadísticas sin almacenarlo todo (magia de memoria).
- Mostrar la diferencia entre “almacenarlo todo” y “transmitirlo todo”.

Poción potenciadora: Los generadores son como una fuente mágica de datos: crean exactamente lo que necesitas, cuando lo necesitas, sin desperdiciar memoria en lo que aún no hace falta.

```
$> python3 ft_data_stream.py
=== Game Data Stream Processor ===

Processing 1000 game events...

Event 1: Player alice (level 5) killed monster
Event 2: Player bob (level 12) found treasure
Event 3: Player charlie (level 8) leveled up
...

=== Stream Analytics ===
Total events processed: 1000
High-level players (10+): 342
Treasure events: 89
Level-up events: 156

Memory usage: Constant (streaming)
Processing time: 0.045 seconds

=== Generator Demonstration ===
Fibonacci sequence (first 10): 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
Prime numbers (first 5): 2, 3, 5, 7, 11
```

¿Cómo permiten los **generadores** un procesamiento eficiente de la memoria? ¿Por qué los **bucles for-in** son perfectos para datos en *streaming*?

Capítulo XI

Ejercicio 6: Alquimista de Datos

	Ejercicio: 6
ft_analytics_dashboard	
Directorio de entrega: *ex*6/	
Archivos a entregar: `ft_analytics_dashboard.py`	
Funciones autorizadas: `List/dict/set comprehensions, len(), print(), sum(), max(), min(), sorted()`	

¡Jefe final! Has dominado todas las estructuras de datos; ahora toca combinarlas en el panel de analítica definitivo. Aquí es donde te certificas como Alquimista de Datos.

Tu misión definitiva: Construir un panel de analíticas usando **comprensiones**, la forma más sólida de Python para transformar datos. Piensa en las comprensiones como hechizos que convierten datos en bruto en información valiosa.

Requerimientos básicos:

- Demostrar **comprensiones de listas** para filtrar y transformar datos.
- Demostrar **comprensiones de diccionarios** para crear mapeos y agrupar datos.
- Demostrar **comprensiones de sets** para encontrar valores únicos.
- Procesar datos de juegos de muestra (puntuaciones, jugadores, logros, etc.).
- Mostrar ejemplos claros de cada tipo de comprensión en acción.

En qué centrarse:

- **Comprensiones de listas**: Filtrar puntuaciones altas, transformar datos, crear nuevas listas.
- **Comprensiones de diccionario**: agrupar jugadores por categoría, contar ocurrencias, crear mapeos.

- **Comprensiones de set**: encontrar jugadores únicos, logros únicos, deduplicar datos.
- Combinar comprensiones con las estructuras de datos de ejercicios anteriores.
- Mantener la sencillez: centrarse en demostrar el dominio de la comprensión, no en crear análisis complejos.

Poción potenciadora: Las comprensiones son como tener una varita mágica: una línea elegante puede transformar conjuntos de datos completos. Es la diferencia entre escribir 10 líneas de bucles frente a 1 línea de pura magia Python.

Ejemplo de resultado (enseña posibles analíticas, ten en cuenta que tu implementación puede variar):

```
$> python3 ft_analytics_dashboard.py
=== Game Analytics Dashboard ===

=== List Comprehension Examples ===
High scorers (>2000): ['alice', 'charlie', 'diana']
Scores doubled: [4600, 3600, 4300, 4100]
Active players: ['alice', 'bob', 'charlie']

=== Dict Comprehension Examples ===
Player scores: {'alice': 2300, 'bob': 1800, 'charlie': 2150}
Score categories: {'high': 3, 'medium': 2, 'low': 1}
Achievement counts: {'alice': 5, 'bob': 3, 'charlie': 7}

=== Set Comprehension Examples ===
Unique players: {'alice', 'bob', 'charlie', 'diana'}
Unique achievements: {'first_kill', 'level_10', 'boss_slayer'}
Active regions: {'north', 'east', 'central'}

=== Combined Analysis ===
Total players: 4
Total unique achievements: 12
Average score: 2062.5
Top performer: alice (2300 points, 5 achievements)
```

Este ejercicio combina todas las estructuras de datos y técnicas que has aprendido. Céntrate en demostrar claramente los **tres tipos de comprensiones**. El ejemplo muestra las posibilidades: tu implementación debe demostrar el dominio de la comprensión, no replicar el resultado exacto.

¡No lo compliques! El objetivo es dominar las comprensiones, no crear un sistema analítico complejo. Utiliza datos de muestra sencillos y codificados (listas, diccionarios, sets) para demostrar cada tipo de comprensión. No lo compliques demasiado: ¡lo importante es la claridad y que domines lo que comprendes!

¿Cómo hacen las **comprensiones** que transformaciones complejas sean legibles? ¿Por qué son esenciales en flujos de trabajo de la ingeniería de datos?

Capítulo XII

Entrega y evaluación

Entrega tu trabajo en tu repositorio `Git` como de costumbre. Solo el trabajo dentro de tu repositorio se evaluará durante la defensa. No dudes en comprobar dos veces los nombres de tus archivos para asegurarte de que son correctos.

Durante la evaluación, se te puede pedir que expliques las elecciones de estructuras de datos, demuestres operaciones con colecciones o amplíes tus sistemas de analítica una con nueva funcionalidad. Asegúrate de comprender los principios detrás de cada estructura de datos.

Debes entregar únicamente los archivos solicitados por el enunciado de este proyecto. Concéntrate en escribir código limpio y bien documentado que demuestre con claridad tu dominio de los tipos de colecciones de Python y de las técnicas de procesamiento de datos.