Retos de Programación en Python: Gestión Inteligente con Diccionarios y Listas

Introducción

Este documento presenta los enunciados de los retos de programación para seis equipos, diseñados para evaluar los conocimientos en Programación Orientada a Objetos (POO) y Funciones y Módulos en Python, con un enfoque en el uso de diccionarios o listas para gestionar datos en una simulación de "Ciudad Inteligente". Cada equipo desarrollará un programa que resuelva un problema específico, utilizando clases con encapsulación, funciones personalizadas, módulos de la librería estándar (`math` o `statistics`), manejo de entrada/salida y manipulación de cadenas. Los retos son equivalentes en complejidad y fomentan la creatividad, la colaboración y la aplicación práctica de los conceptos aprendidos.

1 Enunciados de los Retos

1.1 Equipo 1: Gestor de Rutas de Tráfico

Diseñe una clase `TrafficRoute` para gestionar rutas de tráfico en la ciudad. La clase debe tener atributos para el ID de la ruta (cadena) y un diccionario privado que asocie nombres de intersecciones (cadenas) con tiempos de viaje (en minutos). Implemente un método `add_intersection(name, time)` para agregar una intersección al diccionario y `total_time()` para calcular el tiempo total de la ruta. Cree una función `average_traffic_time(routes)` que use `statistics.mean()` para calcular el tiempo promedio de un conjunto de rutas. Acepte el ID de la ruta y datos de intersecciones mediante `input()`, imprima el diccionario y el tiempo total, y estandarice el ID con `upper()`.

1.2 Equipo 2: Registro de Vehículos Públicos

Cree una clase `PublicVehicle` para registrar vehículos de transporte público. La clase debe incluir atributos para el ID del vehículo (cadena) y una lista privada de paradas visitadas (cadenas). Implemente métodos `visit_stop(stop)` para agregar una parada a la lista y `stop_count()` para contar las paradas. Desarrolle una función `distance_estimate(stops)` que estime la distancia total recorrida usando `math.sqrt()`. Acepte el ID y paradas mediante `input()`, imprima la lista de paradas y la distancia, y use `split()` para procesar una entrada de paradas separada por comas.

1.3 Equipo 3: Analizador de Consumo Eléctrico

Desarrolle una clase `PowerGrid` para analizar el consumo eléctrico de barrios. La clase debe tener atributos para el ID del barrio (cadena) y un diccionario privado que asocie fechas (cadenas) con valores de consumo (en kWh). Implemente métodos `log_consumption(date, value)` para registrar un consumo y `average_consumption()` para calcular el promedio usando `statistics.mean()`. Cree una función `peak_cost(values)` que calcule el costo de picos usando `math.ceil()`. Use `input()` para el ID, fechas y consumos, imprima el diccionario y el promedio, y normalice las fechas con `lower()`.

1.4 Equipo 4: Inventario de Recursos Urbanos

Construya una clase `ResourceInventory` para gestionar recursos urbanos (e.g., bancos, farolas). La clase debe incluir atributos para el ID del inventario (cadena) y una lista privada de recursos (cadenas). Implemente métodos `add_resource(item)` para agregar un recurso y `resource_frequency()` para contar la frecuencia de un recurso específico. Desarrolle una función `resource_priority(count)` que calcule una prioridad usando `math.log()`. Acepte el ID y recursos vía `input()`, imprima la lista y la prioridad, y estandarice los recursos con `replace()`.

1.5 Equipo 5: Sistema de Reservas de Espacios

Diseñe una clase `SpaceReservation` para gestionar reservas de espacios públicos. La clase debe tener atributos para el ID del espacio (cadena) y un diccionario privado que asocie fechas (cadenas) con nombres de usuarios (cadenas). Implemente métodos `reserve(date, user)` para registrar una reserva y `reservation_count()` para contar las reservas. Cree una función `reservation_fee(count)` que calcule una tarifa usando `math.pow()`. Use `input()` para el ID, fechas y usuarios, imprima el diccionario y la tarifa, y use `find()` para verificar si un usuario contiene "VIP".

1.6 Equipo 6: Monitor de Calidad del Aire

Desarrolle una clase `AirQualityMonitor` para monitorear la calidad del aire en zonas urbanas. La clase debe incluir atributos para el ID del monitor (cadena) y una lista privada de lecturas de calidad (en índices numéricos). Implemente métodos `record_reading(value)` para registrar una lectura y `average_quality()` para calcular el promedio usando `statistics.mean()`. Cree una función `pollution_variability(readings)` que calcule la variabilidad con `statistics.stdev()`. Acepte el ID y lecturas vía `input()`, imprima la lista y la variabilidad, y estandarice el ID con `upper()`.

Instrucciones Generales

- **Entregable**: Un archivo `.py` nombrado `equipoX_solución.py` (X es el número del equipo, 1–6).
- **Tiempo**: 2 horas.
- **Criterios de Evaluación** (100 puntos):
  - Funcionalidad (40 puntos): Cumple todos los requisitos y produce la salida correcta.

  - Calidad del código (30 puntos): Organización, legibilidad y uso de encapsulación.
  - Uso de módulos y funciones (20 puntos): Implementación correcta de `math`/`statistics` y funciones personalizadas.
  - Creatividad y presentación (10 puntos): Soluciones innovadoras o explicación clara (opcional, máx. 200 palabras).

- **Restricciones**: Use solo la librería estándar de Python. No se permiten recursos externos durante el reto.