Ejercicio 1: Sistema de Reservas de Vuelos

Requisitos Funcionales:

1. Implementar un sistema de gestión de vuelos utilizando un diccionario anidado con la siguiente estructura:
    - Clave: Código de vuelo (formato "XX-999" donde XX son letras mayúsculas y 999 son dígitos)
    - Valor: Diccionario con origen (string), destino (string), lista de asientos disponibles (lista de strings) y horario (tupla de hora y minutos en formato 24h)
2. Funcionalidades requeridas:
    - Reserva de asientos con validación de disponibilidad
    - Cálculo del porcentaje de ocupación por vuelo
    - Generación de reporte en archivo de texto con los vuelos ordenados por horario
3. Validaciones obligatorias:
    - El código de vuelo debe seguir el patrón especificado
    - Los horarios deben ser válidos (hora: 0-23, minutos: 0-59)
    - Los asientos deben tener formato válido (letra seguida de número)
    - No se permiten reservas duplicadas para el mismo asiento

Requisitos No Funcionales:

- Utilizar el módulo datetime para validación de horarios
- Garantizar complejidad O(1) para operaciones de acceso
- Implementar manejo de excepciones personalizadas

Estructura de Datos:

```
vuelos = {
  "AV-101": {
    "origen": "Lima",
    "destino": "Bogotá",
    "asientos": ["A1", "A2", "B1", "B2"],
    "horario": (15, 30)
  }
}
```

Ejercicio 2: Analizador de Datos Científicos

Requisitos Funcionales:

1. Procesar un dataset de temperaturas globales representado como lista de tuplas con formato (año, mes, temperatura)
2. Funcionalidades requeridas:
    - Cálculo del promedio anual de temperaturas utilizando programación funcional
    - Identificación del mes con temperatura máxima en todo el dataset
    - Validación de integridad del dataset
3. Validaciones obligatorias:
    - Cada año debe contener exactamente 12 registros mensuales
    - Las temperaturas deben estar en rango físico válido (-50.0 a 60.0 °C)
    - Los años deben estar en orden cronológico ascendente

Requisitos No Funcionales:

- Implementar solución utilizando funciones map, filter y reduce
- Documentación completa con docstrings y type hints
- Optimización para manejo de datasets extensos

Estructura de Datos:

```
datos = [
   (2020, 1, 25.3),
   (2020, 2, 26.1),
   # ... resto de datos
]
```

Ejercicio 3: Juego de Rol por Turnos (Versión Estructural)

Requisitos Funcionales:

1. Implementar un sistema de juego por turnos **sin usar POO**, utilizando estructuras de datos básicas
2. Gestionar personajes y sus atributos mediante diccionarios
3. Implementar un sistema de combate por turnos basado en listas
4. Registrar historial de partidas en diccionarios anidados

Estructuras de Datos:

```
# Diccionario principal de personajes
personajes = {
    "merlin": {
        "tipo": "mago",
        "salud": 80,
        "ataque": 30,
        "habilidades": {
            "bola_fuego": 50,
            "curacion": 20
        }
    },
    "gandalf": {
        "tipo": "mago",
        "salud": 90,
        "ataque": 25,
        "habilidades": {
            "rayo": 45,
            "escudo": 15
        }
    }
}
```

```
# Historial de partidas
historial_partidas = {
  "partida_001": {
    "participantes": ["merlin", "gandalf"],
    "turnos": 12,
    "ganador": "merlin",
    "fecha": "2023-11-15"
  }
}
```

Validaciones Obligatorias:

1. **Nombres de personajes:**
    - Solo caracteres alfabéticos y guiones bajos
    - Mínimo 3 caracteres, máximo 15
    - Implementar con re.fullmatch(r'^[a-z_]{3,15}$', nombre)
2. **Atributos numéricos:**
    - Salud y ataque deben ser enteros positivos (1-1000)
    - Daño de habilidades debe ser entero positivo (1-100)
3. **Habilidades:**
    - Verificar existencia antes de usar
    - Nombre de habilidad válido (mismas reglas que nombre de personaje)

Funciones Principales:

```
def crear_personaje(nombre, tipo, salud_base, ataque_base):
  """Valida y crea un nuevo personaje en el diccionario"""
  pass


def ejecutar_turno(jugador_atacante, jugador_defensor, habilidad):
  """Gestiona la mecánica de un turno de combate"""
  pass
```

```
def registrar_partida(participantes, ganador, turnos_totales):
    """Añade una nueva entrada al historial de partidas"""
    pass
```

Requisitos No Funcionales:

1. **Seguridad de datos:**
    - Usar copias profundas (copy.deepcopy) para evitar modificaciones accidentales
    - Validar todos los inputs antes de procesarlos
2. **Eficiencia:**
    - Acceso O(1) a personajes por nombre
    - Operaciones de turno optimizadas para evitar iteraciones innecesarias
3. **Mantenibilidad:**
    - Separar lógica de validación en funciones dedicadas
    - Documentación clara de cada función