Clase 12 – Funciones y Validaciones

Fundamentación

En el desarrollo de software actual, construir código que funcione bajo un "camino feliz" (donde el usuario ingresa exactamente lo que el programador espera) es insuficiente. Los entornos reales exigen programas tolerantes a fallos y con una experiencia de usuario (UX) fluida, incluso ante entradas erróneas.

Hasta este punto, los estudiantes han aprendido a estructurar la lógica mediante funciones y a capturar excepciones de forma aislada. Sin embargo, la saturación de bloques de control de errores (`try/except`) dentro de las funciones de cómputo principales atenta contra la legibilidad, el mantenimiento y la escalabilidad del código.

Esta clase introduce un salto cualitativo en su formación técnica: la **separación de componentes**. Al delegar la captura, validación y limpieza de datos a funciones de control específicas ("escudos"), y el procesamiento matemático a funciones lógicas puras ("motores"), los alumnos no solo asimilan el concepto crítico de la **reutilización de código**, sino que adoptan un estándar profesional de desarrollo que previene fallos críticos del sistema (*crashes*) y mantiene la claridad en la arquitectura del software.

Objetivos

Que los estudiantes logren:

- **Comprender e implementar** el principio de separación de componentes, diferenciando una función de lógica de negocio de una función de validación de datos.
- **Diseñar funciones de validación robustas** utilizando bucles indeterminados (`while True`) combinados con manejo de excepciones (`try/except`) y validaciones lógicas.
- **Reutilizar eficazmente** una misma función modular para sanitizar diferentes entradas de datos dentro del flujo principal de un programa.

"Hasta ahora sabemos crear funciones que calculan cosas y sabemos usar try/except para que el programa no explote si el usuario mete texto en vez de números. El problema es que si metemos todo el try/except dentro de nuestra función principal, el código se vuelve gigante y difícil de leer. Hoy vamos a aprender a separar las tareas: una función exclusiva para limpiar datos y otra función para procesarlos."

Ejercicio principal:

Sistema de Gestión de Notas Escolares

Paso A: La función lógica (El motor)

Esta función se encarga estrictamente de aplicar la regla: calcular el promedio de dos notas. No interactúa con la consola ni le interesa cómo se obtuvieron los datos; solo aplica la fórmula matemática de forma pura.

```
def calcular_promedio_final(nota_trimestre_1, nota_trimestre_2):
    """Calcula el promedio simple entre dos notas."""
    promedio = (nota_trimestre_1 + nota_trimestre_2) / 2
    return promedio
```

Paso B: La función validadora (El escudo)

Se crea una función genérica reutilizable. Aplica un bucle infinito que no se interrumpe hasta que la entrada sea completamente segura y válida (un número flotante que esté estrictamente en la escala escolar de 1 a 10).

```
def pedir_nota_valida(mensaje):
    """Solicita una nota por consola y no avanza hasta que sea un float válido entre 1 y 10."""
    while True:
        try:
            valor = float(input(mensaje))
            if 1 <= valor <= 10:
                return valor  # Corta el bucle y devuelve el dato limpio
            else:
                print("Error: La nota debe estar entre 1 y 10.")
        except ValueError:
            print("Entrada inválida. Por favor, ingresa un número válido (usa punto para decimales).")
```

Paso C: Integración en el flujo principal

El flujo principal queda limpio y ordenado, demostrando visualmente cómo se reutiliza el "escudo" para pedir dos notas de momentos diferentes sin duplicar el código de control de errores.

```
# --- Flujo Principal del Programa ---
print("--- SISTEMA DE CALIFICACIONES ESCOLARES ---")

# Usamos la misma función validadora para ambas notas (Reutilización de código)
nota_1 = pedir_nota_valida("Ingrese la nota del Primer Trimestre: ")
```

```
nota_2 = pedir_nota_valida("Ingrese la nota del Segundo Trimestre: ")


# Pasamos los datos ya limpios y seguros a la función lógica (El motor)
promedio_alumno = calcular_promedio_final(nota_1, nota_2)


print(f"\n El promedio del estudiante es: {promedio_alumno:.2f}")
if promedio_alumno >= 6:
    print("Estado: Aprobado")
else:
    print("Estado: A recuperar")
```

Práctica Autónoma: Desafíos de Programación Relacionados

Opción 1: Registro de Notas de un Curso Completo

- **Consigna:** Diseñar un programa con una función que reciba las notas finales de una cantidad determinada de alumnos de un curso y calcule el promedio general del aula.
- **Blindaje requerido:** El programa debe validar primero que la cantidad de alumnos a registrar sea un número entero mayor a cero. Luego, mediante un bucle, debe pedir la nota de cada alumno validando que esté entre 1 y 10.
- **Restricción de UX:** Si el docente se equivoca al ingresar la nota del alumno número 4, el programa no debe reiniciarse desde el principio; debe mostrar el error y volver a pedir la nota del alumno 4 de forma aislada.

Opción 2: Sistema de Asistencias e Inasistencias

- **Consigna:** Crear un programa que determine si un alumno queda en condición de "Regular" o "Libre" según su porcentaje de asistencia. El límite máximo de días de clase del ciclo es 180.
- **Blindaje requerido:** Una función debe encargarse de solicitar la cantidad de días que asistió el alumno. Debe usar `try/except` para evitar letras y validar lógicamente que los días asistidos estén estrictamente en un rango real (entre 0 y el límite máximo de 180 días). Si introduce un número incoherente como -5 o 200, el sistema debe rechazarlo y solicitarlo nuevamente.

Opción 3: Menú de Gestión del Preceptor (Controlador de Opciones)

- **Consigna:** Crear un programa que simule el panel de un preceptor, donde se pueda elegir entre tres opciones de un menú: `1. Registrar nueva nota`, `2. Ver estadísticas básicas` o `3. Salir del sistema`.

- **Blindaje requerido:** El menú de opciones debe estar completamente blindado. Si el usuario ingresa una opción inexistente (por ejemplo, opción 5) o una letra, el programa debe capturar el error mediante `try/except`, mostrar una advertencia amigable y volver a mostrar el menú completo sin romperse ni cerrarse.

Opción 4: El Contador de Pasos Diario (Fitness)

- **Consigna:** Diseñar un programa con una función que reciba los pasos caminados durante una cantidad de días de la semana y calcule el total acumulado de la semana.
- **Blindaje requerido:** El programa debe validar que la cantidad de días a registrar sea un número entero mayor a cero. Luego, mediante un bucle, debe pedir los pasos de cada día validando que sean enteros válidos. *Restricción de UX:* Si el usuario se equivoca al ingresar los pasos del día 3, el programa no debe reiniciarse desde el principio, debe volver a pedir los pasos del día 3.

Opción 5: El Controlador de Volumen de una Playlist

- **Consigna:** Crear un programa que simule el control de un reproductor de música, donde el usuario puede elegir entre tres opciones de un menú: `1. Subir Volumen`, `2. Bajar Volumen` o `3. Salir`.
- **Blindaje requerido:** Si el usuario elige modificar el volumen, la función lógica debe calcular el nuevo nivel, pero el sistema debe validar que el volumen final se mantenga estrictamente en el rango de **0 a 100** (no puede haber volumen negativo ni mayor a 100). Adicionalmente, el menú de opciones debe estar blindado : si el usuario ingresa una opción inexistente (por ejemplo, opción 5) o una letra, el programa debe capturar el error mediante `try/except`, mostrar un mensaje de advertencia amigable y volver a mostrar el menú sin romperse.