Entrenador de Mecanografía Reactiva

Aplicación de Escritura con Paradigma Reactivo

Cristian Junior Cunurana Calderon

Universidad Nacional del Altiplano
Ingeniería de Sistemas
Lenguajes de Programación

1 de diciembre de 2025

Desarrollado con Python y RxPY

Agenda

1. Introducción
2. Entorno de Desarrollo
3. Tecnologías Utilizadas
4. Programación Reactiva
5. Arquitectura del Sistema
6. Implementación: Lógica Reactiva
7. Implementación: Interfaz Gráfica
8. Visualización y Métricas
9. Conclusiones

Visión General del Proyecto

Objetivo

Desarrollar una aplicación de entrenamiento de mecanografía que utilice el paradigma de **Programación Reactiva** para gestionar el flujo de datos y eventos en tiempo real.

Características Principales:

- ✔ Feedback visual instantáneo.
- ✔ Métricas en tiempo real (WPM, Precisión).
- ✔ Gráficos dinámicos de frecuencia.
- ✔ Gestión de estado inmutable.

Enfoque Técnico:

- </> Separación estricta de Lógica y UI.
- ☰ Uso de Streams para eventos.
- 🔁 Actualizaciones asíncronas.

Infraestructura de Desarrollo

El proyecto ha sido construido y optimizado en un entorno de alto rendimiento.

Especificaciones del Sistema

OS	Fedora Linux 43 (Workstation)
Kernel	Linux 6.17.8-300.fc43.x86_64
CPU	Intel Core i7-12650H (16) @ 4.70 GHz
RAM	16 GiB
DE	GNOME 49.1 (Wayland)
Shell	Bash 5.3.0

Hardware Gráfico

GPU:
NVIDIA GeForce RTX 3050 Mobile

Display:
1920x1080 @ 144 Hz

Stack Tecnológico

- **Lenguaje Core**:
  - **Python 3.x**: Lenguaje base por su versatilidad y ecosistema.
- **Paradigma Reactivo**:
  - **RxPY (Reactive Extensions for Python)**: Biblioteca para componer programas asíncronos y basados en eventos utilizando secuencias observables.
- **Interfaz Gráfica**:
  - **Tkinter**: Biblioteca estándar de GUI para Python.
  - **Matplotlib**: Para la visualización de datos y gráficos estadísticos integrados.

¿Qué es la Programación Reactiva?

Definición

Es un paradigma de programación declarativa concerniente a los flujos de datos (*data streams*) y la propagación del cambio.

Conceptos Clave en el Proyecto:

1. **Observables/Streams**: Secuencias de eventos (teclas, tiempo, señales) a lo largo del tiempo.
2. **Observers/Subscribers**: Componentes que reaccionan a los eventos emitidos.
3. **Operators**: Funciones que transforman, filtran o combinan streams (map, filter, with_latest_from).
4. **State Management**: El estado de la aplicación fluye a través de un *BehaviorSubject*.

Estructura del Proyecto

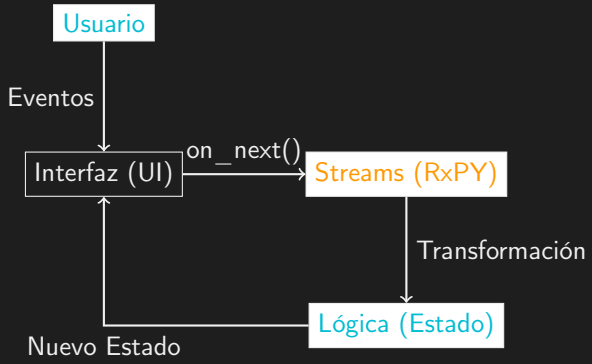
La aplicación sigue una arquitectura modular clara:

`main.py`
: Punto de entrada. Inicializa la aplicación y arranca el loop principal.

`interfaz.py`
: Capa de Vista. Maneja los componentes de Tkinter, captura eventos de teclado y se suscribe a los cambios de estado para actualizar la UI.

`logica.py`
: Capa de Negocio y Estado. Define los streams, el estado reactivo y las funciones puras para la transformación de datos.

Flujo de Datos Unidireccional



Definición de Streams y Estado

En `logica.py`, definimos los canales de comunicación y el estado central.

```python
# Streams para eventos de entrada
tecla_stream = Subject()
borrar_stream = Subject()
palabra_completa_stream = Subject()

# Estado central reactivo (BehaviorSubject)
estado_stream = BehaviorSubject({
    "palabra_actual": "",
    "aciertos": 0,
    "errores": 0,
    "sesion_iniciada": False,
    # ... otros campos
})
```

Tuberías de Procesamiento (Pipes)

La lógica de negocio se aplica transformando los streams de entrada en cambios de estado.

```
# Configuraci n del stream de teclas
tecla_stream.pipe(
    # Obtener el estado actual junto con la tecla
    ops.with_latest_from(estado_stream),

    # Transformar: (tecla, estado) -> nuevo_estado
    ops.map(lambda data: agregar_tecla(data[1], data[0])),

    # Iniciar sesi n si es la primera tecla
    ops.map(lambda estado: iniciar_sesion(estado)),

    # Ignorar si el juego termin
    ops.filter(lambda estado: not estado["finalizado"])
).subscribe(
    # Emitir el nuevo estado resultante
    on_next=lambda nuevo_estado: estado_stream.on_next(nuevo_estado)
)
```

Funciones Puras

Para mantener la predictibilidad, usamos funciones puras que no modifican el estado in-place, sino que retornan uno nuevo.

```python
def validar_palabra(estado):
    """Valida la palabra y retorna nuevo estado"""
    entrada = estado["palabra_actual"].strip()
    esperado = estado["palabras_mostradas"][estado["indice_palabra"]]

    nuevo_estado = estado.copy() # Inmutabilidad

    if entrada == esperado:
        nuevo_estado["aciertos"] += 1
        nuevo_estado["resultado"] = {"mensaje": "Correcto", "color": "#00ff99"}
    else:
        nuevo_estado["errores"] += 1
        nuevo_estado["resultado"] = {"mensaje": "Error", "color": "#ff6666"}

    return nuevo_estado
```

Suscripción a Cambios de Estado

La interfaz es completamente reactiva. No consulta datos, solo reacciona a las emisiones de `estado_stream`.

```
def configurar_suscripciones_reactivas(self):
    # La UI se suscribe al stream de estado
    estado_stream.subscribe(
        on_next=lambda estado: self.actualizar_ui_completa(estado),
        on_error=lambda e: print(f"Error: {e}")
    )

def actualizar_ui_completa(self, estado):
    # Actualizar todos los componentes visuales
    self.ui_components["entrada"].delete(0, tk.END)
    self.ui_components["entrada"].insert(0, estado["palabra_actual"])

    # Actualizar estad sticas
    self.actualizar_estadisticas(estado)

    # Redibujar gr fico
    self._actualizar_grafico(estado)
```

Bindings y Emisión de Eventos

La interfaz captura eventos del usuario y los inyecta en los streams, desacoplando la entrada del procesamiento.

```
def manejar_tecla(event):
    if event.keysym == "space":
        # Emitir evento de palabra completada
        palabra_completa_stream.on_next(True)
        return "break"

    elif len(event.char) == 1:
        # Emitir evento de tecla presionada
        tecla_stream.on_next(event.char)
        return "break"

self.ui_components["entrada"].bind("<Key>", manejar_tecla)
```

Análisis en Tiempo Real

La aplicación utiliza **Matplotlib** embebido en Tkinter para mostrar estadísticas en tiempo real.

Métricas Calculadas:

- **WPM (Words Per Minute)**: Velocidad de escritura normalizada.
- **Precisión**: Porcentaje de aciertos vs errores.
- **Frecuencia de Teclas**: Histograma dinámico que muestra qué teclas se presionan más.

Gráfico

[Visualización de Barras]
Frecuencia de caracteres
actualizada en cada pulsación.

Conclusiones y Futuro

Logros

- Implementación exitosa del patrón reactivo en una aplicación de escritorio.
- Desacoplamiento total entre la lógica de negocio y la interfaz de usuario.
- Experiencia de usuario fluida con feedback visual inmediato.

Trabajo Futuro

- Persistencia de datos (base de datos de puntuaciones).
- Modos de juego adicionales (contrarreloj, práctica de código).
- Soporte multijugador mediante websockets.

¡Gracias por su atención!

github.com/cjelgamer
ccunurana.netsec@gmail.com