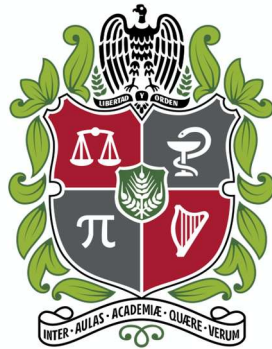


UNIVERSIDAD NACIONAL DE COLOMBIA

FACULTAD DE INGENIERÍA



TEMA:

PROGRAMACIÓN REACTIVA

PRESENTADO A:

GUSTAVO ADOLFO MOJICA PERDIGÓN

PRESENTADO POR:

JOSÉ YECID JIMÉNEZ BUITRAGO

jyjimenezb@unal.edu.co

Fecha de entrega: 18/07/2025

Nota Aclaratoria:

Este documento se presenta como trabajo individual. Por motivos personales (incompatibilidad de horarios en la parte laboral y especialmente, cuidado de un familiar convaleciente) no me fue posible coordinar con el grupo originalmente asignado (Grupo 15).

Las capturas de pantalla, texto de los códigos para ser testeados, etc. se incluyen como apéndices.

1. Introducción

La programación reactiva es un paradigma de la Programación que se usa para manejar datos y eventos que cambian con el tiempo; ejemplo: cuando el usuario mueve el mouse y el programa responde con alguna acción. Este tipo de programación se usa cada vez más actualmente, porque las aplicaciones modernas, sitios web, etc. deben responder en tiempo real; por eso es importante entenderla.

En este paper se explicará qué es este paradigma, origen, sus diferencias con otros paradigmas y se presentará un ejemplo práctico en python.

También se hará una publicación en github. (Repositorio)

2. Parte Teórica.

2.1. Definición y Fundamentos. La programación reactiva es

un paradigma que se basa en reaccionar (ejecutar una acción) a un determinado cambio que ocurra en el sistema, por ejemplo: cuando el usuario ejecuta una acción (da click ó pulsa una tecla) ó cuando se envía o recibe un dato a través de un puerto de comunicación.

Este paradigma es útil cuando se trabaja con interfaces gráficas, sensores, señales, streamings, etc. porque hay cambios constantes y el sistema tiene que estar "pendiente" de estos cambios y en consecuencia ejecutar una acción.

Los principios fundamentales de la programación reactiva son:

- Flujo de datos → El sistema procesa los datos en tiempo real
- Eventos asíncronos → Los eventos no ocurren en orden fijo
- Propagación de cambios → Los cambios se propagan
- No bloqueante → El programa no se detiene si espera un evento

2.2. Comparación con otros paradigmas:

Paradigma	Descripción	Ventajas	Desventajas
Imperativo	Programación paso a paso.	Fácil de entender (al inicio)	Difícil con tareas complejas
Declarativo	Dice qué hacer pero no cómo hacerlo	Facilidad para usar con bases de datos	Difícil en cosas específicas
Reactivo	Basado en flujos y eventos que cambian	Útil en apps interactivas, sensores	Difícil de depurar y entender (al principio)
POO.	Basado en clases y objetos que contienen datos y comportamientos.	Facilita ordenar el código y reutilizarlo.	Curva de aprendizaje amplia.

2.3. Historia y Evolución:

Desde hace décadas los lenguajes de programación han usado elementos como "Event Listeners" para hacer que el programa o aplicación reaccione a algún cambio.

Con el auge de aplicaciones más interactivas (celulares, aplicaciones web) el paradigma se empezó a desarrollar más formalmente.

En la década de 2010 surgieron librerías como RxJava, RxJS, Reactor, etc. que dieron forma al enfoque reactivo, en Python está RxPy.

Actualmente muchos sistemas modernos usan el paradigma: desde apps móviles hasta robots que deben recibir datos en tiempo real.

2.4 Casos de Uso:

Ejemplos donde se usa la programación reactiva:

- Apps Web donde hay un flujo constante de datos
(Ej: Un web chat)
- Sistema de control de sensores en tiempo real.
(Ej: Robótica)
- Apps móviles fáciles ó que responden al movimiento
(Ej: Cuando se gira el celular para que la pantalla quede de forma horizontal).
- Procesamiento de streams de video ó audio.
- Automatización industrial.
(Ej: Cadenas de proceso ó montaje industrial)

2.5. Lenguajes de Programación Asociados:

Se pueden usar muchos lenguajes para hacer programación reactiva, sin embargo algunos lenguajes tienen mejor soporte para este Paradigma:

- Java (RxJava, Reactor)
- Javascript (RxJS)
- C# (Reactive Extensions)
- Kotlin (Flow y Coroutines)
- Scala (Akka Streams)
- Python (RxPY)

Código de Ejemplo en Python:

```
import rx
```

```
from rx import operators as ops
```

```
source = rx.of("Alpha", "Beta", "Gamma", "Delta", "Epsilon")
```

```
disp = source.subscribe(lambda value: print(f"Received {value}"), lambda ex: print(ex), lambda: print('completed'))
```

Fuente: <https://medium.com/@michamarszaek/reactive-programming-in-python-2af1495c7922>

3 Parte Práctica.

El siguiente código en python toma una lista de números y elige los números mayores a 3 y los imprime usando un flujo de datos:

Primero debe instalarse RxPY:

```
pip install rx
```

Código:

```
# Se importa RxPY
from rx import from_iterable
from rx import operators as ops

# Lista a usar de ejemplo
numeros = [1, 2, 3, 4, 5]

# Se crea el flujo a partir de la lista
flujo = from_iterable(numeros)

# Filtro que solo deja pasar números > 3
flujo_filtrado = flujo.pipe(
    ops.filter(lambda x: x > 3)
)

# Suscripción para cada vez que un número > 3 se imprima
flujo_filtrado.subscribe(
    on_next=lambda n: print("Recibí el número: ", n),
    on_completed=lambda: print("Ya no hay más números.")
)
```

3.1. Explicación del código

- Se usa `from_iterable` para hacer un flujo de datos desde una lista.
- Se usa `pipe()` con `filter` para filtrar los mayores que 3
- Se suscribe al flujo para reaccionar a los datos que pasan el filtro
- `on_next` imprime cada número recibido
- `on_completed` imprime cuando ya no quedan más números

El código es simple y corto y demuestra los principios de:

- flujo de datos (`from_iterable`)
- transformación (`filter`)
- Reacción a eventos (`subscribe`)

3.2. Desafíos y Consideraciones:

La programación reactiva es útil y moderna pero no es fácil de aprender y/o entender en un principio

La primera dificultad es que el programa "no hace nada" hasta que no se hace la suscripción al flujo en el código.

Otro problema recurrente es entender el orden de las cosas; algunas veces no es claro cuál es el resultado que se va a imprimir, por ejemplo, si se coloca map antes de filter, el resultado puede cambiar y resultar confuso.

Otra fuente de errores ó confusión ocurre con las funciones (por ejemplo en la lambda del map) lo cual puede romper todo el flujo.

Estos errores pueden ser difíciles de detectar en la depuración.

Comparación con otros paradigmas:

La comparación más recurrente es con el imperativo (for, if, listas, etc) y se siente muy distinto.

En el imperativo se tiene todo el control, al programa se le dice qué hacer paso a paso. En la programación reactiva se le debe decir al sistema qué debe pasar cuando ocurra algo y eso puede ser difícil de visualizar.

Comparación de códigos en ambos paradigmas:

Por ejemplo, este código usa un ciclo for para imprimir el contenido de una variable si cumple una condición if:

```
for numero in numeros:  
    if numero > 3:  
        print(numero)
```

En cambio, bajo el paradigma reactivo, el código luce así:

```
from rx import from_iterable  
from rx import operators as ops  
from_iterable(numeros).pipe(  
    ops.filter(lambda x: x > 3)  
).subscribe(lambda x: print(x))
```

La segunda forma es más compacta pero más difícil de leer, especialmente para desarrolladores novatos o que hasta ahora están aprendiendo. También, si hay errores, es bastante más difícil encontrarlos.

Reflexión:

Este estilo o paradigma puede ser muy útil cuando se trabaja con un flujo constante y cambiante de datos (Ej: sensores, robótica, etc) pero para cosas más sencillas (recorrer listas, hacer cálculos, leer archivos, etc), resulta mejor el estilo "normal".

4. Conclusiones:

Al realizar este trabajo aprendí sobre la programación reactiva, que es un paradigma diferente al usado de forma cotidiana (imperativo); este estilo de programación se basa en reaccionar a datos que cambian en función del tiempo u otras variables.

A través del ejemplo práctico se entendió cómo funciona un flujo de datos, cómo aplicar transformaciones (filtros) y de qué manera el programa puede reaccionar de forma automática al flujo de datos (subscribe); las funciones como pipe, lambda, etc; permiten que el código sea más modular y limpio.

Desde la perspectiva de la ingeniería, este paradigma tiene muchas aplicaciones prácticas (sensores que transmiten datos en tiempo real, sistemas de monitoreo, robots que deben reaccionar rápidamente ante cambios de su entorno, etc).

Resumiendo, aunque la programación reactiva no es sencilla al principio, tiene un enorme potencial y vale la pena seguir explorándola; al combinarla con otros paradigmas, permite resolver problemas complejos de forma eficiente.

Referencias Bibliográficas

Reactive programming in Python

(<https://medium.com/@michamarszaek/reactive-programming-in-python-2af1495c7922>)

¿Qué es la programación reactiva y cómo funciona?

(<https://ed.team/blog/que-es-la-programacion-reactiva-y-como-funciona>)

Functional reactive programming

(https://en.wikipedia.org/wiki/Functional_reactive_programming#:~:text=reactive%20programming%20%20,56)

Programación Reactiva, un Paradigma necesaria para los tiempos actuales

(<https://blog.tenea.com/programacion-reactiva-un-paradigma-necesario-para-los-tiempos-actuales/>)

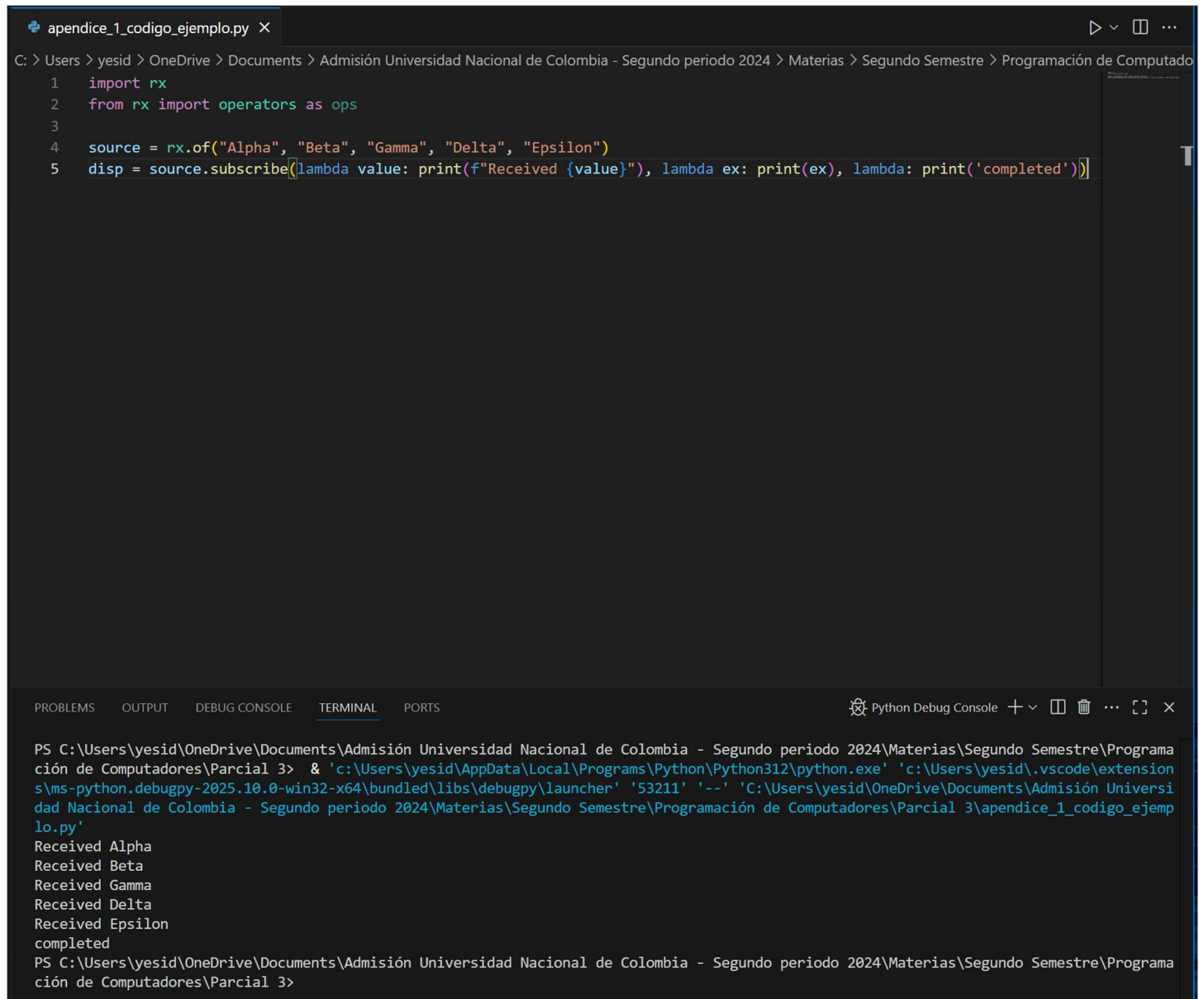
Apéndice 1.

```
import rx

from rx import operators as ops


source = rx.of("Alpha", "Beta", "Gamma", "Delta", "Epsilon")

disp = source.subscribe(lambda value: print(f"Received {value}"), lambda ex:
print(ex), lambda: print('completed'))
```



The screenshot shows a VS Code editor window with a file named `apendice_1_codigo_ejemplo.py`. The code in the editor is as follows:

```
1 import rx
2 from rx import operators as ops
3
4 source = rx.of("Alpha", "Beta", "Gamma", "Delta", "Epsilon")
5 disp = source.subscribe(lambda value: print(f"Received {value}"), lambda ex: print(ex), lambda: print('completed'))
```

Below the editor, the `TERMINAL` panel is active, showing the output of the script. The output is:

```
PS C:\Users\yesid\OneDrive\Documents\Admisión Universidad Nacional de Colombia - Segundo periodo 2024\Materias\Segundo Semestre\Programación de Computadores\Parcial 3> & 'c:\Users\yesid\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\yesid\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53211' '--' 'C:\Users\yesid\OneDrive\Documents\Admisión Universidad Nacional de Colombia - Segundo periodo 2024\Materias\Segundo Semestre\Programación de Computadores\Parcial 3\apendice_1_codigo_ejemplo.py'
Received Alpha
Received Beta
Received Gamma
Received Delta
Received Epsilon
completed
PS C:\Users\yesid\OneDrive\Documents\Admisión Universidad Nacional de Colombia - Segundo periodo 2024\Materias\Segundo Semestre\Programación de Computadores\Parcial 3>
```

Apéndice 2.

```
# Se importa RxPY
from rx import from_iterable
from rx import operators as ops

# Lista a usar de ejemplo
numeros = [1, 2, 3, 4, 5]

# Se crea el flujo a partir de la lista
flujo = from_iterable(numeros)

# Filtro que solo deja pasar números > 3
flujo_filtrado = flujo.pipe(
    ops.filter(lambda x: x > 3)
)

# Suscripción para cada vez que un número > 3 se imprima
flujo_filtrado.subscribe(
    on_next=lambda n: print("Recibí el número:", n),
    on_completed=lambda: print("Ya no hay más números.")
)
```


OneDrive > Documents > Admisión Universidad Nacional de Colombia - Segundo periodo 2024 > Materias > Segundo Semestre > Programación de Computadores > Par

```

1  # Se importa RxPY
2  from rx import from_iterable
3  from rx import operators as ops
4
5  # Lista a usar de ejemplo
6  numeros = [1, 2, 3, 4, 5]
7
8  # Se crea el flujo a partir de la lista
9  flujo = from_iterable(numeros)
10
11 # Filtro que solo deja pasar números > 3
12 flujo_filtrado = flujo.pipe(
13     ops.filter(lambda x: x > 3)
14 )
15
16 # Suscripción para cada vez que un número > 3 se imprima
17 flujo_filtrado.subscribe(
18     on_next=lambda n: print("Recibí el número:", n),
19     on_completed=lambda: print("Ya no hay más números.")
20 )

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python Debug Console + ▼ □ 🗑 ... [] X

```

PS C:\Users\yesid\OneDrive\Documents\Admisión Universidad Nacional de Colombia - Segundo periodo 2024\Materias\Segundo Semestre
\Programación de Computadores\Parcial 3> & 'c:\Users\yesid\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\yesid
\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53698' '--' 'C:\Users\yesid\OneDrive\
Documents\Admisión Universidad Nacional de Colombia - Segundo periodo 2024\Materias\Segundo Semestre\Programación de Computador
es\Parcial 3\apendice_2_parte_practica.py'

```

Recibí el número: 4

Recibí el número: 5

Ya no hay más números.

```

PS C:\Users\yesid\OneDrive\Documents\Admisión Universidad Nacional de Colombia - Segundo periodo 2024\Materias\Segundo Semestre
\Programación de Computadores\Parcial 3>

```

Apéndice 3.

Enlace repositorio GitHub:

https://github.com/lord-yasha/parcial_3_programacion_reactiva