Escuela Politécnica Nacional

Nombre: Francisco Ulloa
Fecha: Quito, 28 de enero de 2026
Tema: Módulo Logging
Repositorio:
https://github.com/Fu5CHAR/Metodos_numericos_2025B_Ulloa-Francisco/tree/main

¿Qué es el modulo Logging?

El módulo logging de Python es una biblioteca estándar integrada para registrar eventos, errores y mensajes informativos durante la ejecución de una aplicación. Permite rastrear el comportamiento del programa, facilitando la depuración (debugging) y auditoría mediante la creación de archivos de log personalizados con diferentes niveles de gravedad.

Los niveles incluyen:

- **NONSET (nivel 0):** Cuando se establece en un logger, indica que los loggers antecesores deben ser consultados para determinar el nivel efectivo. Si el resultado sigue siendo NOTSET, se registrarán todos los eventos.
- **DEBUG (nivel 10):** Información detallada, normalmente sólo de interés para un desarrollador que intenta diagnosticar un problema.
- **INFO (nivel 20):** Confirmación de que todo funciona según lo previsto.
- **WARNING (nivel 30):** Una indicación de que ha ocurrido algo inesperado o de que podría producirse un problema en un futuro próximo.
- **ERROR (nivel 40):** Debido a un problema más grave, el software no ha podido realizar alguna función.
- **CRITICAL (nivel 50):** Un error grave, que indica que es posible que el propio programa no pueda seguir ejecutándose.

¿Qué son los Handlers?

Los handlers (manejadores) del módulo logging de Python son componentes encargados de dirigir los registros (logs) generados por un Logger a sus destinos finales, como la consola, archivos, o correos electrónicos. Actúan como el puente entre el mensaje de registro y el medio de salida, permitiendo filtrar y dar formato a los mensajes.

Aspectos clave de los Handlers:

- **Destinos múltiples:** Un mismo logger puede tener múltiples handlers (por ejemplo, registrar errores en un archivo y mostrar avisos en la consola).
- **Filtrado de Nivel:** Cada handler puede tener su propio nivel de severidad (DEBUG, INFO, ERROR), filtrando mensajes independientemente del nivel del logger.
- **Formatters:** Permiten asignar un formato específico a la salida de cada manejador.

Tipos comunes:

- **StreamHandler:** Envía logs a la consola (stdout/stderr).
- **FileHandler:** Guarda los logs en un archivo. RotatingFileHandler: Gestiona archivos de log, rotándolos al alcanzar cierto tamaño.
- **SMTPHandler:** Envía logs por correo electrónico.

En resumen, los handlers definen dónde y cómo se almacenan o visualizan los mensajes de log en una aplicación Python.

Ejemplos del uso de Logging

In [30]:
```
from custom_logger import CustomLogger
logger = CustomLogger(
    name="logging_modificado",
    log_file="salida.log",
    telegram_token="8194345808:AAF0VQmIYF8xkBV916DbYPg7tzvzmVxAqdk",
    telegram_chat_id="7324662557"
).get_logger()
```

In [31]:
```
logger.debug('prueba debug')
```

```
2026-01-26 21:43:59 | DEBUG    | 1970062199.py:1 | prueba debug
DEBUG:logging_modificado:prueba debug
DEBUG:logging_modificado:prueba debug
```

In [32]:
```
logger.warning('prueba warning')
```

```
2026-01-26 21:43:59 | WARNING  | 2881943780.py:1 | prueba warning
WARNING:logging_modificado:prueba warning
WARNING:logging_modificado:prueba warning
```

In [33]:
```
logger.info('mensaje en archivo')
```

```
2026-01-26 21:43:59 | INFO     | 71576706.py:1 | mensaje en archivo
INFO:logging_modificado:mensaje en archivo
INFO:logging_modificado:mensaje en archivo
```

Según la configuración utilizada los mensajes de nivel INFO también se imprimen en el archivo salida.log

In [34]:
```
logger.error('prueba error')
```

```
2026-01-26 21:43:59 | ERROR    | 1655888829.py:1 | prueba error
ERROR:logging_modificado:prueba error
```

In [35]:
```
logger.critical('prueba critical')
```

```
2026-01-26 21:44:01 | CRITICAL | 2813385120.py:1 | prueba critical
CRITICAL:logging_modificado:prueba critical
```

Según la configuración utilizada los mensajes de nivel ERROR Y CRITICAL también se imprimen en el archivo salida.log y son enviados a un chat de Telegram