



PYTHON

File Processing

Logging - Basics logging facility for Python

FILE PROCESSING

Logging - Basics logging facility for Python

FILE PROCESSING

- La biblioteca estándar de Python proporciona el **modulo logging** para registrar logs.
 - `import logging`
- Permite dejar logs en diferentes ubicaciones: ficheros, flujos de salida o en servicios externos.
- El método **getLogger** proporciona un objeto de la clase **Logger**. Invocado sin argumentos, proporciona el denominado **root logger**. Si se le indica un nombre (con la notación de los módulos, utilizando el punto como separador) proporciona *loggers* específicos.

```
import logging

logger = logging.getLogger()
hello_logger = logging.getLogger('hello')
hello_world_logger = logging.getLogger('hello.world')
recommended_logger = logging.getLogger(__name__)
```

- Una buena práctica es obtener el Logger con el nombre del módulo en el que se encuentra (`__name__`)

FILE PROCESSING

- El **modulo logging** proporciona diferentes niveles de criticidad para los log que pueden ser ampliados. Cada nivel tiene un número, siendo el más alto el correspondiente al más crítico.
- La clase **Logger** proporciona métodos para escribir logs relacionados con estos niveles: **critical**, **error**, **warning**, **info** y **debug**.
- En la configuración por defecto, los errores de nivel INFO y DEBUG no se muestran.

Level name	Value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

FILE PROCESSING

- Ejemplo:

```
import logging
```

```
logging.basicConfig()
```

```
logger = logging.getLogger()
```

```
logger.critical('Your CRITICAL message')
```

```
logger.error('Your ERROR message')
```

```
logger.warning('Your WARNING message')
```

```
logger.info('Your INFO message')
```

```
logger.debug('Your DEBUG message')
```



```
CRITICAL:root:Your CRITICAL message  
ERROR:root:Your ERROR message  
WARNING:root:Your WARNING message
```

- El método **basicConfig** establece una configuración por defecto en la que aparecen los logs con el siguiente formato:
 - NIVEL:nombre del logger:Mensaje
- EJERCICIO: Crea varios logger con distintos nombres y comprueba el resultado al escribir sobre ellos

FILE PROCESSING

- El método **setLevel** permite indicar el nivel mínimo de logs que debe escribir.
- Por defecto, el nivel mínimo es WARNING, dejando fuera a los niveles DEBUG e INFO.
- Si el logger se crea con nombre (un logger que no es *root logger*) el nivel asignado es NOTSET. Tomará el nivel del *logger* más próximo en la jerarquía. Si este es también NOTSET seguirá escalando hasta obtener el nivel del *root logger*.

FILE PROCESSING

- Ejemplo de asignación de nivel delegado.

```
import logging

logging.basicConfig()
logger = logging.getLogger()
mi_logger = logging.getLogger('mi.logger')

logger.setLevel(logging.DEBUG)
# Ejecutar con la línea comentada y descomentada
# mi_logger.setLevel(logging.ERROR)

logger.critical('Your CRITICAL message')
logger.error('Your ERROR message')
logger.warning('Your WARNING message')
logger.info('Your INFO message')
logger.debug('Your DEBUG message')

mi_logger.critical('Your CRITICAL message')
mi_logger.error('Your ERROR message')
mi_logger.warning('Your WARNING message')
mi_logger.info('Your INFO message')
mi_logger.debug('Your DEBUG message')
```

FILE PROCESSING

- Ejemplo de asignación de nivel delegado con jerarquía.

```
import logging

logging.basicConfig()

logger = logging.getLogger()
logger_nivel_1 = logging.getLogger('nivel1')
logger_nivel_2 = logging.getLogger('nivel1.nivel2')

logger.setLevel(logging.ERROR)
# Ejecutar con esta línea comentada y descomentada
#logger_nivel_1.setLevel(logging.WARNING)

logger.critical('Your CRITICAL message')
logger.error('Your ERROR message')
logger.warning('Your WARNING message')
logger.info('Your INFO message')
logger.debug('Your DEBUG message')

logger_nivel_2.critical('Your CRITICAL message')
logger_nivel_2.error('Your ERROR message')
logger_nivel_2.warning('Your WARNING message')
logger_nivel_2.info('Your INFO message')
logger_nivel_2.debug('Your DEBUG message')
```


FILE PROCESSING

- El método **basicConfig** utilizado sin argumentos proporciona una configuración predeterminada y crea un objeto de la clase **StreamHandler** que muestra los registros por consola.
- Algunos de los argumentos que admite **basicConfig** son:
 - **level** → Nivel mínimo de los logs. **Afecta a los logger que no tengan nivel asignado.**
 - **filename** → Fichero al que se van a volar los logs. En lugar de generar un objeto **StreamHandler**, **genera un objeto FileHanler.**
 - **filemode** → Modo de apertura del fichero de log: 'a' para añadir.

```
import logging

logging.basicConfig(level=logging.CRITICAL, filename='prod.log',
filemode='a')

logger = logging.getLogger()

logger.critical('Your CRITICAL message')
logger.error('Your ERROR message')
logger.warning('Your WARNING message')
logger.info('Your INFO message')
logger.debug('Your DEBUG message')
```

FILE PROCESSING

- El formato por defecto de los logs utilizando basicConfig es:
 - NIVEL:nombre del logger:Mensaje
- Este formato se puede cambiar con la propiedad **format** de basicConfig, que admite los atributos del objeto **LogRecord** (se crea automáticamente).
- Algunos atributos de LogRecord:
 - **name**: Nombre del logger que creó el registro.
 - **level**: Nivel de severidad (ej. logging.INFO).
 - **message**: Mensaje del registro (puede contener marcadores de formato).
 - **asctime**: Marca de tiempo
 - **pathname**: Ruta del archivo donde se generó el registro.
 - **lineno**: Número de línea del código donde ocurrió el evento.
 - **args**: Argumentos usados en el mensaje (si los hay): Ejemplo: logger.critical('Error %s', 'fatal')
 - **exc_info**: Información de la excepción si se está registrando un error. El log debe escribirse dentro de un except e incluir el argumento exc_info=True

```
import logging
FORMAT = '%(name)s:%(levelname)s:%(asctime)s:%(message)s'
logging.basicConfig(level=logging.CRITICAL, filename='prod.log', filemode='a', format=FORMAT)
logger = logging.getLogger()
```

FILE PROCESSING

- Personalización:
 - **formatter**
 - **handler**

```
import logging

# Creación del logger
logger = logging.getLogger(__name__)

# Creación del formateador
FORMAT = '%(name)s:%(levelname)s:%(asctime)s:%(message)s'
formatter = logging.Formatter(FORMAT)

# Creación del handler
handler = logging.FileHandler('prod.log', mode='a')

# Asignación del nivel de criticidad al handler
handler.setLevel(logging.CRITICAL)

# Asignación del formateador al handler
handler.setFormatter(formatter)

# Asignación del handler al logger
logger.addHandler(handler)

# Uso
logger.critical('Your CRITICAL message')
logger.error('Your ERROR message')
logger.warning('Your WARNING message')
logger.info('Your INFO message')
logger.debug('Your DEBUG message')
```