Logging

Cronograma

- Módulo Logging
- Configuración Básica
- Formateando la salida:
  - Logging de variables
  - Stack Traces
- Clases y Funciones
- Handlers
- Archivos de configuración.

Introducción al Logging

- Guardar registros o logear es una buena práctica de programación para registrar el flujo de un programa.
- Ayuda a entender qué fue lo que sucedió ante un eventual error o situación no contemplada.
- Un correcto logging permite guardar diferentes valores de las variables: usuario, ip, url que se solicitó en un determinado tiempo.
- No solo permite analizar errores también permite analizar la performance de la aplicación.

Módulo Logging

- Python provee un sistema de registro de información como parte de la librería estándar.
- Fácil de integrar:

  import logging
- Al importar el módulo podemos utilizar las funciones del módulo para registrar información.
- Existen 5 niveles de severidad :
  1. DEBUG
  2. INFO
  3. WARNING
  4. ERROR
  5. CRITICAL

Módulo Logging

```
1  import logging
2
3  logging.debug('Mensaje en modo Debug')
4  logging.info('Mensaje en modo Info')
5  logging.warning('Mensaje en modo Warning')
6  logging.error('Mensaje en modo Error')
7  logging.critical('Mensaje en modo Critico')
8
```

- Probemos como se ven las salidas
- Formato de salida estándar:

  Severidad:Módulo Logging:Mensaje
- Podemos agregar más información: Timestamp, Línea, Cardinal, etc.
- ¿Por qué no se ven todos los mensajes?
- Por defecto solamente se muestran mensajes del tipo WARNING o con mayor severidad.

Configuración

- El método basicConfig(**kwargs) permite configurar el logging.
- Parámetros:
  a. **level** : Severidad del módulo raíz.
  b. **filename**: Archivo de salida.
  c. **filemode**: Modo de apertura del archivo de logging. Por defecto es a (append)
  d. **format**: Formato de mensaje de salida.
- El parámetro **level** nos permite configurar cual es la severidad mínima que queremos registrar.

```
12  logging.basicConfig(level=logging.DEBUG)
13  logging.debug('Este mensaje ahora si se muestra')
```

Salida a archivo

Podemos guardar los mensajes en archivo.

Debemos utilizar los parámetros: **filename y filemode**

```
logging.basicConfig(filename='log_file.log', filemode='w', format='%(name)s - %(levelname)s - %(message)s')
logging.warning('Mensaje que se almacena en el log')
```

Salida:

root - WARNING - Mensaje que se almacena en el log

- El mensaje se guarda en el arghico log_file.log y no se muestra en consola.
- Formato de archivo “w” indica que el archivo se abre en modo escritura.
- Cada vez que se ejecuta la aplicación se reescribe el archivo de logs.

Formateando la salida.

Aparte de escribir mensajes de texto al log también podemos pasar atributos generales del programa. Por ejemplo el id del proceso.

```
16  logging.basicConfig(format='%(process)d-%(levelname)s-%(message)s')
17  logging.warning('Mansaje con el procceso agregado')
```

Existen varios atributos más del sistema que podemos pasarle a nuestro log. Algunas son:

Atributo	Formato	Descripción
asctime	%(asctime)s	Hora de creación del mensaje.
created	%(created)f	Timestamp del mensaje
filename	%(filename)s	Nombre del archivo
levelname	%(levelname)s	Severidad del mensaje
module	%(module)s	Nombre del módulo
process	%(process)d	Número del proceso.
message	%(message)s	Mensaje de salida

Registro de variables

- Podemos incluir información dinámica de aplicación.
- Al mensaje de salida le podemos pasar variables a mostrar.
- Se puede utilizar los f-string para generar los mensajes.

```
22  logging.basicConfig(format='%(asctime)s - %(levelname)s - %(message)s')
23  user = "rperez"
24  logging.warning(f'Usuario logeado: {user}')
```

Stack Trace

- El módulo de logging también permite capturar los stack trace de la pila de la aplicación.
- Es útil cuando catcheamos una excepción y queremos mostrar la pila de la llamada
- Debemos de pasar el parámetro exc_info en True
- También existe el método **logging.exception("Mensaje")** que hace exactamente lo mismo

```
26  logging.basicConfig(format='%(levelname)s - %(message)s')
27
28  a = 5
29  b = 0
30
31  try:
32      c = a / b
33  except Exception as e:
34      logging.error("Exception occurred", exc_info=True)
```

Clases y Funciones

- Hasta ahora utilizamos el módulo de logging por defecto o root.
- Es buena práctica definir tu propio módulo de loggeo utilizando la clase Logger.
- Esto es bueno para utilizar el mismo Logger en todos nuestros módulos.
- Nuestro módulo de loggeo debe tener:
  - Logger: Instancia de la clase logging definida. Tendrá toda la información de cómo se debe registrar el mensaje.
  - LogRecord: Evento con el mensaje y su formato.
  - Handler: Envía el objeto LogRecord para que tome la decisión de mostrar el mensaje. Tenemos los FileHandler, StreamHandler, SMTPHandler y HTTPHandler.
  - Formatter: Especificación de cómo mostrar un mensaje.

Clases y Funciones (II)

```
2  import logging
3
4  logger = logging.getLogger('example_logger')
5  logger.warning('Warning Message')
6
```

- Aquí estamos creando un logger con nombre “example_logger”
- Este logger que pedimos a la clase Logging es diferente a la instancia root que veníamos utilizando.
- Esta instancia que creamos no se puede configurar con el método **basicConfig()**
- Debemos configurarlo con Handlers y Formatters:
- Es recomendable que utilizamos módulos de loggeo definidos por nosotros y cuando los creemos le pasemos el valor de __name__.
- Esto hace que cuando generemos un evento aparezca el nombre de la clase de donde se invoco.

Handlers

- Los creamos para definir nuestro propios Loggers y enviar los eventos a diferentes lugares.
- Los handlers permiten enviar los mensajes a la entrada estándar, a un archivo, por http o por SMTP.
- El logger que creemos puede tener más de un Handler. Por ejemplo que envíe el mensaje a un archivo y por correo.
- También podemos definir el nivel de severidad mínimo que queremos registrar.

Ejemplo

```python
import logging

# Creamos el logger
logger = logging.getLogger(__name__)

# Creamos los handler
c_handler = logging.StreamHandler()
f_handler = logging.FileHandler('file.log')
c_handler.setLevel(logging.WARNING)
f_handler.setLevel(logging.ERROR)

# Creamos los formatos de salida de los mensajes y se los asociamos a los handler.
c_format = logging.Formatter('%(name)s - %(levelname)s - %(message)s')
f_format = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
c_handler.setFormatter(c_format)
f_handler.setFormatter(f_format)

# Asociamos el handler al logger
logger.addHandler(c_handler)
logger.addHandler(f_handler)

# Genermos el evento
logger.warning('This is a warning')
logger.error('This is an error')
```

Archivo de configuración

- Podemos armar la configuración del logger utilizando un archivo externo o un diccionario.
- La información se carga con fileConfig() o dictConfig() respectivamente.
- Por ejemplo si utilizamos un archivo de configuración lo debemos cargar con:

```
1  import logging
2  import logging.config
3
4  logging.config.fileConfig(fname='file.conf', disable_existing_loggers=False)
5
6  # Get the logger specified in the file
7  logger = logging.getLogger(__name__)
8
9  logger.debug('This is a debug message')
10
```

Archivo de configuración

```
[loggers]
keys=root,sampleLogger

[handlers]
keys=consoleHandler

[formatters]
keys=sampleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_sampleLogger]
level=DEBUG
handlers=consoleHandler
qualname=sampleLogger
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=sampleFormatter
args=(sys.stdout,)

[formatter_sampleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
```