Creado por:

Isabel Maniega

Logging en Python

La biblioteca estándar de Python proporciona un módulo útil llamado `logging` para registrar eventos que ocurren en la aplicación. Los registros se utilizan con mayor frecuencia para encontrar la causa de un error. De forma predeterminada, Python y sus módulos proporcionan muchos registros que le informan sobre las causas de los errores. Sin embargo, es una buena práctica crear sus propios registros que pueden ser útiles para usted o para otros programadores.

Un ejemplo de uso de sus propios registros puede ser cualquier sistema de Internet. Cuando los usuarios visitan su sitio, puede registrar información sobre los navegadores que utilizan. Si algo sale mal, podrá determinar en qué navegadores se está produciendo el problema.

En Python, puede almacenar registros en diferentes lugares. La mayoría de las veces, se encuentra en forma de archivo, pero también puede ser un flujo de salida o incluso un servicio externo. Para comenzar a registrar, necesitamos importar el módulo apropiado:

```
import logging
```

En esta parte del curso, aprenderá a crear registros utilizando el módulo `logging`. Vea lo que ofrece este módulo y comience a usarlo para convertirse en un mejor programador.

El objeto Logger

Una aplicación puede tener varios registradores creados tanto por nosotros como por los programadores de los módulos. Si su aplicación es simple, como en el ejemplo a continuación, puede usar el registrador raíz. Para ello, llame a la función getLogger sin proporcionar un nombre. El registrador raíz está en el punto más alto de la jerarquía. Su lugar en la jerarquía se asigna en función de los nombres pasados a la función `getLogger`.

Los nombres de los registradores son similares a los nombres de los módulos de Python en los que se utiliza el separador de puntos. Su formato es el siguiente:

- **hello** – crea un registrador que es un hijo del registrador raíz;
- **hello.world** – crea un registrador que es un hijo del registrador hello.

Si desea realizar otra anidación, simplemente utilice el separador de puntos.

La función `getLogger` devuelve un objeto `Logger`. Veamos el código de ejemplo en el editor. Allí encontraremos las formas de obtener el objeto `Logger`, tanto con nombre como sin él.

Recomendamos llamar a la función `getLogger` con el argumento `__name__`, que se reemplaza por el nombre del módulo actual. Esto le permite especificar fácilmente la fuente del mensaje registrado.

NOTA: Varias llamadas a la función `getLogger` con el mismo nombre siempre devolverán el mismo objeto.

In [1]:
```
import logging

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

```
<Logger hello.world (WARNING)>
```

Niveles de registro

El objeto `Logger` permite crear registros con distintos niveles de registro que ayudan a distinguir entre registros menos importantes y aquellos que informan un error grave. De forma predeterminada, se definen los siguientes niveles de registro:

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

Cada nivel tiene un `nombre` y un valor `numérico`. También puedes definir tu propio nivel, pero los que ofrece el módulo `logging` son suficientes. El objeto Logger tiene métodos que establecen el nivel de registro por ti. Observa el ejemplo en el editor.

Resultado:

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

Todos los métodos anteriores requieren que proporciones un mensaje que será visible en los registros. El formato de registro predeterminado incluye el nivel, el nombre del registrador y el mensaje que hayas definido. Ten en cuenta que todos estos valores

están separados por dos puntos. Más adelante en este curso, aprenderás a cambiar el formato predeterminado.

Probablemente te estés preguntando por qué no se muestran los mensajes con niveles INFO y DEBUG. Esto se debe a la configuración predeterminada, de la que hablaremos en un momento.

NOTA: El método `basicConfig` se analizará más adelante en el curso. Por ahora, recuerda que es responsable de la configuración básica del registro.

In [2]:
```
import logging

logging.basicConfig()

logger = logging.getLogger('Hola')

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:Hola:Your CRITICAL message
ERROR:Hola:Your ERROR message
WARNING:Hola:Your WARNING message
```

El método setLevel

El registrador raíz tiene el nivel de registro establecido en `WARNING`. Esto significa que los mensajes en los niveles `INFO` o `DEBUG` no se procesan.

A veces, es posible que desee cambiar este comportamiento, especialmente si crea su propio registrador. Para ello, debe pasar un nivel de registro al método `setLevel`. Vea cómo lo hacemos en el editor.

Resultado:

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

La configuración del nivel `DEBUG` hace que se registren los mensajes con este nivel o uno superior. Vale la pena mencionar que los registradores creados con el argumento de nombre tienen el nivel `NOTSET` establecido de forma predeterminada. En este caso, su nivel de registro se establece en función de los niveles principales, comenzando por el principal más cercano al registrador raíz.

Si el principal más cercano tiene un nivel establecido en `NOTSET`, el nivel del registrador se establece en función de los niveles de los principales subsiguientes en la jerarquía. La configuración de nivel finaliza si un principal tiene un nivel distinto de

`NOTSET`. Si ninguno de los principales visitados tiene un nivel distinto de `NOTSET`, entonces se procesarán todos los mensajes independientemente de su nivel.

In [3]:
```
import logging

logging.basicConfig()

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

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
INFO:root:Your INFO message
DEBUG:root:Your DEBUG message
```

Configuración básica (parte 1)

Como mencionamos antes, la configuración básica de registro se realiza mediante el método `basicConfig`. Al llamar al método `basicConfig` (sin especificar ningún argumento), se crea un objeto `StreamHandler` que procesa los registros y luego los muestra en la consola.

El objeto `StreamHandler` es creado por el objeto `Formatter` predeterminado responsable del formato del registro. Como recordatorio, el formato predeterminado consta del nombre del nivel, el nombre del registrador y el mensaje definido. Finalmente, el controlador recién creado se agrega al registrador raíz. Más adelante, aprenderá a crear su propio controlador y formateador.

En los ejemplos anteriores, llamamos al método `basicConfig` sin ningún argumento. Usando el método `basicConfig`, puedes cambiar el nivel de registro (de la misma manera que usando el método `setLevel`) e incluso la ubicación de los registros. Observa el ejemplo en el editor.

Resultado en el archivo `prod.log`:

```
CRITICAL:root:Tu mensaje CRÍTICO
```

En el ejemplo, el método `basicConfig` toma tres argumentos. El primero es el nivel de registro igual a `CRITICAL`, lo que significa que solo se procesarán los mensajes con este nivel.

Pasar un nombre de archivo al segundo argumento crea un objeto `FileHandler` (en lugar de un objeto `StreamHandler`). Como probablemente hayas notado, los registros ya no aparecen en la consola. Después de configurar el argumento `filename`, todos los registros se dirigirán al archivo especificado.

Además, pasar el último argumento `filemode` con el valor `'a'` (este es el modo predeterminado) significa que se agregarán nuevos registros a este archivo. Si desea cambiar este modo, puede usar otros modos que sean análogos a los utilizados en la función incorporada `open`.

Estos no son todos los argumentos que puede tomar el método `basicConfig`. ¿Está listo para otra dosis de conocimiento? ¡Sigamos adelante!

NOTA: El método `basicConfig` cambia la configuración del registrador raíz y sus hijos que no tienen su propio controlador definido.

In [6]:
```
import logging

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

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
INFO:root:Your INFO message
DEBUG:root:Your DEBUG message
```

Configuración básica (parte 2)

El método `basicConfig` presentado anteriormente también se puede utilizar para cambiar el formato de registro predeterminado. Esto se hace utilizando el argumento `format`, que se puede definir utilizando cualquier carácter o atributo del objeto `LogRecord`. Vamos a explicarlo con el ejemplo en el editor.

Resultado en el archivo prod.log:

```
root:CRITICAL:2019-10-10 17:16:46,293:Su mensaje CRÍTICO
```

El formato que definimos se crea combinando los atributos del objeto `LogRecord` separados por dos puntos. El objeto `LogRecord` es creado automáticamente por el registrador durante el registro. Contiene muchos atributos, como el nombre del `logger`, el `logging level` o incluso el número de línea en el que se llama al método `logging`. Puede encontrar una lista completa de todos los atributos disponibles aquí [https://docs.python.org/3/library/logging.html#logrecord-attributes].

En nuestro ejemplo, usamos los siguientes atributos:

%(name)s – este patrón será reemplazado por el nombre del registrador que llama al método de registro. En nuestro caso, es el registrador raíz;

%(levelname)s – este patrón será reemplazado por el nivel de inicio de sesión establecido. En nuestro caso, este es el nivel `CRITICAL`;

%(asctime)s – este patrón será reemplazado por un formato de fecha legible para humanos que indica cuándo se creó el objeto `LogRecord`. El valor decimal se expresa en milisegundos;

%(message)s – este patrón será reemplazado por el mensaje definido. En nuestro caso, es `'Su mensaje CRÍTICO'`.

En general, el esquema para usar el argumento del objeto `LogRecord` en el argumento de formato se ve así:

```
(LOG_RECORD_ATTRIBUTE_NAME)s
```

In [7]:
```
import logging

FORMAT = '%(name)s:%(levelname)s:%(asctime)s:%(message)s'

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

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
INFO:root:Your INFO message
DEBUG:root:Your DEBUG message
```

Su primer controlador

Cada registrador puede guardar registros en diferentes ubicaciones y en diferentes formatos. Para ello, debe definir su propio controlador y formateador.

En la mayoría de los casos, querrá guardar sus registros en un archivo. El módulo de registro tiene la clase `FileHandler`, que facilita esta tarea. Al crear un objeto `FileHandler`, debe pasar un nombre de archivo donde se guardarán los registros.

Además, puede pasar un modo de archivo con el argumento mode, por ejemplo, `mode='a'`. En el siguiente paso, debe establecer el nivel de registro que procesará el controlador. De forma predeterminada, el controlador recién creado se establece en el nivel `NOTSET`. Puede cambiarlo utilizando el método `setLevel`. En el ejemplo del editor, hemos establecido el nivel `CRITICAL`.

Por último, debes agregar el manejador creado a tu registrador usando el método `addHandler`.

Resultado en el archivo `prod.log`:

```
Tu mensaje CRÍTICO
```

Si revisas el archivo `prod.log`, verás que solo el mensaje se guarda allí. ¿Sabes lo que olvidamos? Tu manejador no ha creado un formateador. Aprenderás cómo hacer esto en un momento.

NOTA: Cada registrador puede tener varios manejadores agregados. Un manejador puede guardar registros en un archivo, mientras que otro puede enviarlos a un servicio externo. Para procesar mensajes con un nivel inferior a `WARNING` por manejadores agregados, es necesario establecer este umbral de nivel en el registrador raíz.

In [8]:
```
import logging

logger = logging.getLogger(__name__)

handler = logging.FileHandler('prod.log', mode='w')
handler.setLevel(logging.CRITICAL)

logger.addHandler(handler)

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:__main__:Your CRITICAL message
ERROR:__main__:Your ERROR message
WARNING:__main__:Your WARNING message
INFO:__main__:Your INFO message
DEBUG:__main__:Your DEBUG message
```

Tu primer formateador

¡Felicitaciones! Acabas de crear tu primer controlador. Solo falta el formateador, pero no te preocupes. Son solo dos pasos. Observa el ejemplo en el editor.

Resultado en el archivo `prod.log`:

```
__main__:CRITICAL:2019-10-10 20:40:05,119:Tu mensaje
CRÍTICO
```

En el primer paso, debes crear un objeto `Formatter` pasando el formato que has definido a su constructor. En el ejemplo, usamos el formato definido en uno de los ejemplos anteriores.

El siguiente paso es configurar el formateador en el objeto `handler`. Esto se hace usando el método `setFormatter`. Después de hacer esto, puedes analizar tus registros en el archivo `prod.log`.

In [9]:
```
import logging

FORMAT = '%(name)s:%(levelname)s:%(asctime)s:%(message)s'
```

```
logger = logging.getLogger(__name__)

handler = logging.FileHandler('prod.log', mode='w')
handler.setLevel(logging.CRITICAL)

formatter = logging.Formatter(FORMAT)
handler.setFormatter(formatter)

logger.addHandler(handler)

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:__main__:Your CRITICAL message
ERROR:__main__:Your ERROR message
WARNING:__main__:Your WARNING message
INFO:__main__:Your INFO message
DEBUG:__main__:Your DEBUG message
```

Creado por:

Isabel Maniega