

ASIMOV

Logging

Logging é o processo de registrar eventos que ocorrem ao longo da execução de um software. É uma ferramenta importantíssima para qualquer programador, permitindo a ele obter um profundo conhecimento sobre o funcionamento de seu código, localizar bugs e otimizar seu script. Há muitas formas de executar logging de seu código e aqui, aprenderemos a como usar o módulo padrão do Python.

Uso básico

A importação do módulo é bem simples, e não precisa de instalações adicionais.

```
In [13]: # Importando a biblioteca
import logging
```

O módulo fornece uma forma simples para que os aplicativos configurem diferentes manipuladores de log e uma maneira de enviar mensagens de log para esses manipuladores. Isso permite um setup flexível que pode lidar com muitos casos de uso diferentes.

Para registrarmos nossos logs, primeiramente instanciamos um objeto responsável por manipular estes registros através do método `logging.getLogger()`. Através destes objetos, podemos criar formas para nosso programa lidar diferentes tipos de logs.

```
In [14]: log = logging.getLogger("meu-logger")
log.info("Hello, world") # Loga uma informação do tipo INFO
```

Níveis de logging

Nem todas as mensagens de log são iguais. Ao definir um nível de registro em Python usando o módulo padrão, você está informando à biblioteca que deseja lidar com todos os eventos desse nível em diante. Se você definir o nível de log para **INFO**, ele incluirá as mensagens INFO, WARNING, ERROR e CRITICAL. As mensagens NOTSET e DEBUG não serão incluídas aqui.

Nível	Valor numérico
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10

Nível	Valor numérico
NOTSET	0

Podemos registrar eventos de cada um destes níveis, simplesmente chamando a função `log.nivel()` apropriada, por exemplo:

```
In [15]: log.critical("Registra um log do nível info")
log.error("Registra um log do nível error")
log.warning("Registra um log do nível warning")
log.info("Registra um log do nível info")
log.debug("Registra um log do nível debug")
```

```
Registra um log do nível info
Registra um log do nível error
Registra um log do nível warning
```

Configuração básica

Para a configuração básica, você pode usar o método `basicConfig(**kwargs)` para configurar o logging:

Alguns dos parâmetros comumente usados para o `basicConfig()`:

- `level`: O logger raiz será definido para o nível de loggin especificado.
- `filename`: especifica o nome do arquivo.
- `filemode`: Se o nome do arquivo for fornecido, o arquivo é aberto neste modo. O padrão é `a`, o que significa anexar.
- `format`: este é o formato da mensagem de registro.

Usando o parâmetro `level`, você pode definir o nível de mensagens de log que deseja registrar. Isso pode ser feito passando uma das constantes disponíveis na classe, e isso permitiria que todas as chamadas de registro em ou acima desse nível fossem registradas. Aqui está um exemplo:

```
In [16]: import logging

logging.basicConfig(level=logging.DEBUG)
logging.debug('This will get logged')
```

```
DEBUG:root:This will get logged
```

Todos os eventos no nível `DEBUG` ou acima serão agora registrados.

Da mesma forma, para registrar em um arquivo ao invés do console, podemos usar os parametros `filename` e `filemode`, e você pode decidir o formato da mensagem usando `format`. O exemplo a seguir mostra o uso de todos os três:

```
In [17]: import logging

logging.basicConfig(filename='app.log', filemode='w', format='%(name)s - %(levelname)s',
logging.warning('This will get logged to a file')
```

```
WARNING:root:This will get logged to a file
```

A mensagem será semelhante a esta, mas será gravada em um arquivo denominado `app.log`

ao invés de registrar no console. O `filemode` é definido como `w`, o que significa que o arquivo de log é aberto no “modo de gravação” cada vez que `basicConfig()` é chamado, e cada execução do programa irá reescrever o arquivo. A configuração padrão para o modo de arquivo é `a`, que é anexado.

Você pode personalizar o logger raiz ainda mais usando mais parâmetros para `basicConfig()`, que podem ser encontrados [aqui](#).

Deve-se notar que chamar `basicConfig()` para configurar o logger root funciona apenas se o logger root não tiver sido configurado antes. Basicamente, esta função só pode ser chamada uma vez.

Usar os métodos `debug()`, `info()`, `warning()`, `error()` e `critical()` também chamam o `basicConfig()` (sem argumentos) automaticamente se não tiver sido chamado antes. Isso significa que após a primeira vez que uma das funções acima é chamada, você não pode mais configurar o logger root porque eles teriam chamado a função `basicConfig()` internamente.

Para resetar o `basicConfig`, podemos usar:

```
In [8]: def reset_log():
        for handler in logging.root.handlers[:]:
            logging.root.removeHandler(handler)
```

```
In [18]: reset_log()
```

Formatando o Output

Embora seja possível passar qualquer variável para ser representada como uma string de seu programa como uma mensagem para seus logs, existem alguns elementos básicos que já fazem parte do `LogRecord` e podem ser facilmente adicionados ao formato de saída. Se quiser registrar o ID do processo junto com o nível e a mensagem, você pode fazer algo assim:

```
In [19]: import logging

logging.basicConfig(format='%(process)d-%(levelname)s-%(message)s')
logging.warning('This is a Warning')
```

```
13848-WARNING-This is a Warning
```

A variável `format` pode receber uma string com atributos `LogRecord` em qualquer arranjo que você quiser. A lista completa de atributos disponíveis pode ser encontrada [aqui](#).

Aqui está outro exemplo onde você pode adicionar a data e hora em

```
In [20]: reset_log()

logging.basicConfig(format='%(asctime)s - %(message)s', level=logging.INFO)
logging.info('Admin logged in')
```

```
2021-06-17 14:20:59,269 - Admin logged in
```

Registro de dados variáveis

Na maioria dos casos, você deseja incluir informações dinâmicas de seu aplicativo nos logs. Você viu que os métodos de registro aceitam uma string como argumento e pode parecer natural formatar uma string com dados de uma variável em uma linha separada e passá-la para o método de registro. Mas isso pode ser feito diretamente usando uma string formatada para a mensagem e anexando os dados variáveis como argumentos. Aqui está um exemplo:

In [23]:

```
reset_log()

name = 'John'

logging.error('%s raised an error', name)
```

```
ERROR:root:John raised an error
```

Capturando erros completos

O módulo de registro também permite capturar os erros completos em um aplicativo. As informações de exceção podem ser capturadas se o parâmetro `exc_info` for passado como `True` e as funções de registro forem chamadas assim:

In [24]:

```
reset_log()

a = 5
b = 0

try:
    c = a / b
except Exception as e:
    logging.error("Exception occurred", exc_info=True)
```

```
ERROR:root:Exception occurred
Traceback (most recent call last):
  File "<ipython-input-24-ed24265e9133>", line 7, in <module>
    c = a / b
ZeroDivisionError: division by zero
```