

Live de Python #48

Criando logs de aplicações



Nome:

Eduardo Mendes

Instituição:

Unicamp / Diebold Nixdorf

Contatos:

{facebook, github,
instagram, linkedin,
telegram, twitter}/dunossauro

Roteiro

- Que raios são logs?
- Qual a importância de usar logs?
- Logando na prática
 - Introdução a biblioteca logging
 - Levels do logger



Que raios são logs?

O log é um meio de rastrear eventos que acontecem quando algum software é executado. O desenvolvedor do software adiciona chamadas ao seu código para indicar que certos eventos ocorreram.

Os eventos também têm uma importância que o desenvolvedor atribui ao evento; a importância também pode ser chamada de nível ou gravidade.



Que raios são logs?

Imagine que exista alguma coisa em algum determinado sistema. Uma tela de login, por exemplo.

Imagine que você precisa saber sempre quem logou em um determinado horário. Por exemplo, as pessoas que logaram no sistema para saber o resultado de um jogo de futebol no horário de almoço?



Qual a importância de usar logs?

Os log fornecem uma explicação simples de algum evento “qualquer” no sistema. Ou seja, você consegue visualizar estados de determinadas execuções do sistema. **Lembra quando você quer saber de algo e coloca um monte de prints?** Os logs além de separarem em níveis diferentes certos tipos de mensagem, podem ser usados em casos especiais. Até mesmo em auditorias de sistemas.



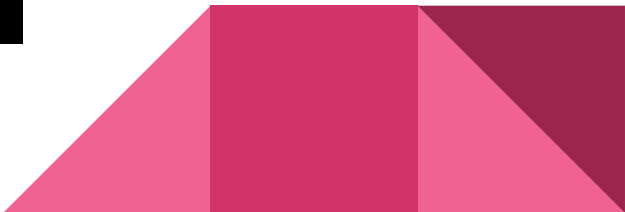
Níveis do logger

Level	Quando usar isso
DEBUG	Informações mais detalhadas, quando estamos buscando problemas
INFO	Confirmar que as coisas estão funcionando como esperado
WARNING	Informação de que algo inesperado aconteceu (mas tudo funciona bem)
ERROR	Quando algo inesperado ocorre e o programa não consegue executar algo
CRITICAL	Um erro grave que impediu o sistema de executar algo

Níveis do logger

Por default, o nível, ou level, como preferir chamar é WARNING. Ou seja, se usarmos a definição normal do log, os eventos com menor prioridade que WARNING não serão exibidos.

```
In [1]: import logging  
  
In [2]: logging.debug('debug')  
  
In [3]: logging.info('info')
```



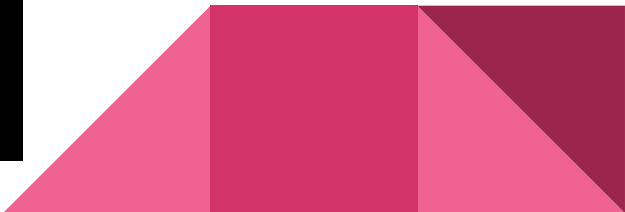
Níveis do logger

Por default, o nível, ou level, como preferir chamar é WARNING. Ou seja, se usarmos a definição normal do log, os eventos com menor prioridade que WARNING não serão exibidos.

```
In [4]: logging.warning('warning')  
WARNING:root:warning
```

```
In [5]: logging.critical('critical')  
CRITICAL:root:critical
```


```
In [6]: logging.error('error')  
ERROR:root:error
```



Escrevendo o log em um arquivo

```
• import logging
  logging.basicConfig(filename='exemplo.log',
                      level=logging.DEBUG)

  logging.debug('debug')
  logging.info('info')
  logging.warning('warning')
```

 exemplo.log

DEBUG:root:debug

INFO:root:info

WARNING:root:warning

Personalizando o log

O log pode ser “formatado de maneira extremamente simples”. Existe uma tabela na documentação (linkada abaixo) que mostra todos os “atributos” para facilitar a escrita do log.

Atributo	formato	Descrição
asctime	%(asctime)s	Mostra a hora do log (d/m/a-h:m:s)
filename	%(filename)s	O arquivo em que a chamada foi feita
levelname	%(levelname)s	Nível do log
message	%(message)s	Mensagem logada

<https://docs.python.org/3/library/logging.html#logrecord-attributes>


Personalizando o log

```
import logging

log_format = '%(asctime)s:%(filename)s:%(levelname)s:%(message)s'

logging.basicConfig(filename='exemplo_3.log',
                    level=logging.DEBUG,
                    format=log_format)
```

```
2018-04-29 19:49:56,782:exemplo_3.py:INFO:Olá Marilene
2018-04-29 19:49:56,782:exemplo_3.py:DEBUG:Hoje a noite
2018-04-29 19:49:56,782:exemplo_3.py:CRITICAL:vinho, Tainha, e ...
```



Tá, sabemos o
básico e agora?

Nível de abstração

Logger

Level

Handler

Formater

Filter

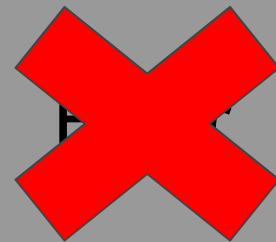
Nível de abstração

Logger

Level

Handler

Formater



Logger

A classe de principal da lib logging, **logger** tem o seu trabalho dividido em três partes.

1. Ela emite métodos para que os loggers possam escrever em *runtime*
2. Determina quais mensagens devem ser ser filtradas (por default usa os níveis)
3. Gerencia os loggers enviando mensagens para os **handlers** interessados em receber aquelas mensagens.

Os métodos mais usados são divididos em duas categorias.



Logger

Configuração

- `setLevel`
- `addHandler`
- `removeHandler`
- `addFilter`
- `removeFilter`

Você não precisa chamar eles, a biblioteca faz isso pra você.

Envio de mensagens

- `debug`
- `info`
- `warning`
- `error`
- `critical`
- `log` (interface genérica)
- `exception`

logging.getLogger(qualname='root')

É uma interface que devolve um logger específico que foi configurado. Caso o mesmo nome seja invocado repetidamente, o mesmo logger será retornado.

```
import logging

log_format = '%(asctime)s:%(filename)s:%(levelname)s:%(message)s'

logging.basicConfig(filename='exemplo_3.log',
                    level=logging.DEBUG,
                    format=log_format)

logger = logging.getLogger()
```

Handler

Handlers são os manipuladores para o output dos loggers, ou seja. Um logger por ter mais de um handler e um handler pode estar em mais de um logger. Eles são independentes.

Imagine que haja uma vontade de enviar logs para tela e para arquivos ao mesmo tempo? Um logger pode dois handlers associados.

Do mesmo modo que vários loggers podem compartilhar o mesmo handler que escreve em um arquivo.

[Alguns handlers prontos na biblioteca](#)



Handler

Caso você precise de um manipulador diferente dos presentes na biblioteca padrão, você pode herdar de `logging.Handler` e criar o seu próprio, ela é uma ABC.

[Alguns handlers prontos na biblioteca](#)

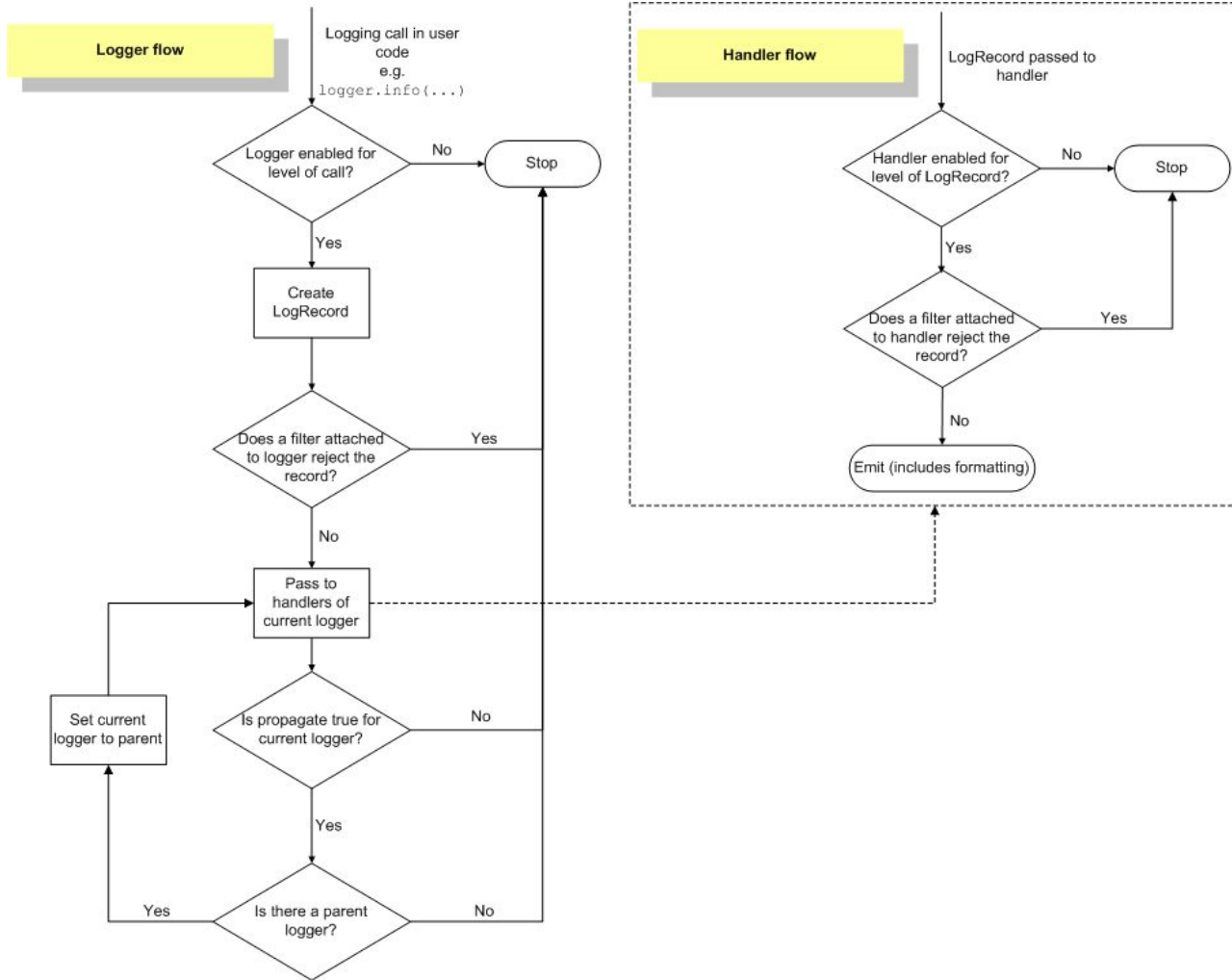


Formatter

Os formataadores são o final da cadeia, ele é chamado pelo handler para formatar as mensagens.

```
Formatter.__init__(fmt=None, datefmt=None, style='%)
```

- `fmt`: formato da mensagem
- `datefmt`: formato de como a hora será apresentado
- `style`: formatador que dará início ao “itens formatados”



Exemplo 1:

Usando código para definir um logger completo

Exemplo 2:

Usando arquivos de configuração para definir um logger completo

<https://docs.python.org/3.6/library/logging.config.html#logging-config-api>