Criando logs para aplicações

Live de Python #198

Roteiro

1. Afinal, o que são logs?

Para que eles servem?

2. Criando logs de maneira simples

Usando a biblioteca logging e seus níveis

3. Entendendo o mecanismo dos logs

Filtros, Handlers e formatação de logs

4. Conhecendo o loguru

Uma simplificação aos logs embutidos

5. Logs no mundo externo

Uma integração básica com Sentry

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Acássio Anjos, Ademar Peixoto, A Earth, Alexandre Harano, Alexandre Souza, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Ana Padovan, Andre Azevedo, André Rocha, Aquiles Coutinho, Arnaldo Turque, Ayrton Freeman, Bloquearsites Farewall, Bruno Barcellos, Bruno Freitas, Bruno Guizi, Bruno Oliveira, Bruno Ramos, Caio Nascimento, César Almeida, Christiano Morais, Clara Battesini, Cleber Santos, Daniel Haas, Danilo Segura, Dartz Dartz, David Kwast, Delton Porfiro, Dhyeives Rodovalho, Diego Guimarães, Dilenon Delfino, Donivaldo Sarzi, Douglas Bastos, Douglas Braga, Douglas Martins, Douglas Zickuhr, Emerson Rafael, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Evandro Avellar, Fabiano Gomes, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Flávio Meira, Flavkaze Flavkaze, Franklin Silva, Gabriel Barbosa, Gabriel Simonetto, Geandreson Costa, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Gustavo Dettenborn, Heitor Fernandes, Henrique Junqueira, Igor Taconi, Ismael Ventura, Israel Gomes, Italo Silva, Jair Andrade, Janael Pinheiro, João Lugão, Johnny Tardin, Jonatas Leon, Jonatas Oliveira, Jônatas Silva, Jorge Plautz, Jose Mazolini, Juan Gutierrez, Juliana Machado, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Botassio, Leandro Miranda, Leonardo Cruz, Leonardo Mello, Leonardo Nazareth, Lucas Adorno, Lucas Mello, Lucas Mendes, Lucas Oliveira, Lucas Polo, Lucas Praciano, Lucas Teixeira, Lucas Valino, Luciano Silva, Luciano Teixeira, Luiz Junior, Luiz Lima, Maiquel Leonel, Marcelino Pinheiro, Marcelo Matte, Márcio Martignoni, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Marina Passos, Mario Deus, Matheus Silva, Matheus Vian, Murilo Andrade, Murilo Cunha, Murilo Viana, Natan Cervinski, Nicolas Teodosio, Osvaldo Neto, Patricia Minamizawa, Patrick Brito, Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, Priscila Santos, Rafael Lopes, Rafael Romão, Ramayana Menezes, Reinaldo Silva, Renan Moura, Renato Veirich, Richard Nixon, Riverfount Riverfount, Rodrigo Ferreira, Rodrigo Freire, Rodrigo Junior, Rodrigo Vaccari, Rogério Sousa, Ronaldo Silva, Rui Jr, Samanta Cicilia, Sara Selis, Thiago Araujo, Thiago Borges, Thiago Bueno, Thiago Curvelo, Thiago Moraes, Tony Dias, Victor Wildner, Vinícius Bastos, Vinicius Figueiredo, Vítor Gomes, Vitor Luz, Vlademir Souza, Vladimir Lemos, Wellington Abreu, Wesley Mendes, William Alves, Willian Lopes, Wilson Neto, Yury Barros

Obrigado você

Salvando a sua vida!

Logs

O que são logs?

Log é uma expressão usada para descrever o processo de **registro de eventos relevantes** em um sistema [1].

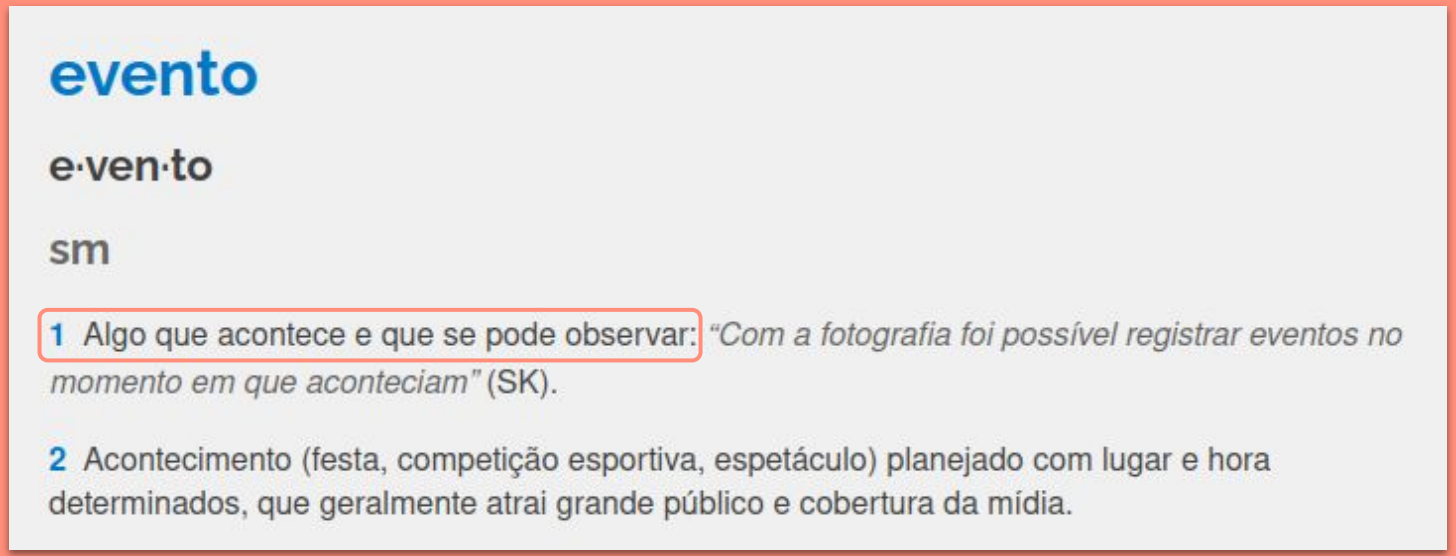
Mas o que seriam **eventos**?

1. https://pt.wikipedia.org/wiki/Log_de_dados

O que são logs?

Log é uma expressão usada para descrever o processo de **registro de eventos relevantes** em um sistema [1].

Mas o que seriam **eventos**?



evento

e·ven·to

sm

1 Algo que acontece e que se pode observar: *"Com a fotografia foi possível registrar eventos no momento em que aconteciam"* (SK).

2 Acontecimento (festa, competição esportiva, espetáculo) planejado com lugar e hora determinados, que geralmente atrai grande público e cobertura da mídia.

michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/evento/

1. https://pt.wikipedia.org/wiki/Log_de_dados

Exemplo de eventos

- Quando alguém entra em um sistema
  - **log**in
  - **log**on
- Quando alguém sai em um sistema
  - **log**off
  - **log**out
- Quando uma determinada ação acontece no nosso sitema
  - Chamar uma determinada função
  - Um determinado erro acontece na aplicação
  - ...

Log é uma maneira de observar

Log é um registro de algo que **já aconteceu**, está no passado.

Usamos os logs para saber o que aconteceu com a aplicação, isso pode nos ajudar a observar como a aplicação está se saindo em produção.

```python
from logging import error


def divisão(dividendo, divisor):
    try:
        return dividendo / divisor
    except ZeroDivisionError:
        error('Tentaram dividir por zero!')
        return 0
```

Log é uma maneira de observar

Log é um registro de algo que **já aconteceu**, está no passado.

Usamos os logs para saber o que aconteceu com a aplicação, isso pode nos ajudar a observar como a aplicação está se saindo em produção.

```python
from logging import error


def divisão(dividendo, divisor):
    try:
        return dividendo / divisor
    except ZeroDivisionError:
        error('Tentaram dividir por zero!')
        return 0
```

```
$ python exemplo_00.py
ERROR:root:Tentaram dividir por zero!
```

Mas não é melhor usar o debugger?

Os logs, embora sejam menos poderosos que as ferramentas de debug, eles atuam em outra camada.

A ideia é logar em ambientes onde não podemos debugar.

Por exemplo, uma aplicação na nuvem não pode ser debugada. Podemos logar todos os erros e tentar debugar na nossa máquina usando os parâmtreos que recebemos lá.

Além de **entender como a aplicação está rodando**, podemos ler somente os logs.

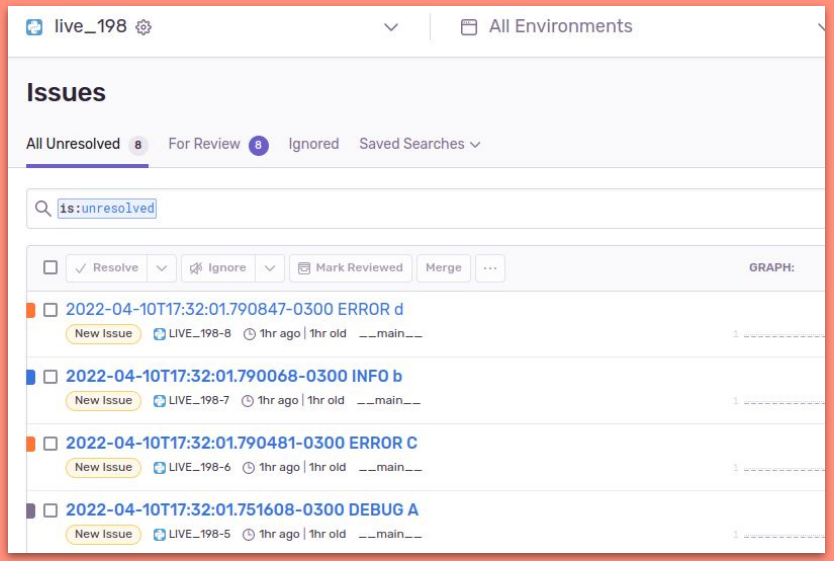
Como os logs são guardados?

Geralmente os logs são jogados na saída padrão, assim como o print. Ou em um arquivo de texto. Com isso, alguma aplicação externa faz a leitura.

E no lugar de ficar lendo linhas e mais linhas em um arquivo, essa aplicação para logs nos ajuda a filtrar, armazenar e ver o que importa.

Exemplo de aplicações desse tipo:

- logstash
- **sentry***
- datadog
- newrelic
- ....



Entendendo como logar

O básico

Como logar com python?

O python tem uma biblioteca nativa para trabalhar com logs. A biblioteca **logging**.

```python
from logging import error, critical, warning

# Qualquer evento
# ...
warning('Algo inesperado aconteceu')
error('Um erro aconteceu!')
critical('Algo critico aconteceu!')
```

Como logar com python?

O python tem uma biblioteca nativa para trabalhar com logs. A biblioteca **logging**.

```python
from logging import error, critical, warning

# Qualquer evento
# ...
warning('Algo inesperado
error('Um erro aconteceu
critical('Algo critico a
```

```
$ python exemplo_00.py

WARNING:root:Um erro aconteceu!
ERROR:root:Um erro aconteceu!
CRITICAL:root:Algo critico aconteceu!
```

Warning? Error? Critial?

Os logs costumam ser geridos por **níveis** ou **gravidade**. Sempre que registramos um evento, queremos saber se é só um aviso "passei por aqui" ou se a aplicação parou de funcionar por alguns minutos.

Os níveis são divididos em:

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

E você pode escolher a partir de qual nível quer monitorar

A configuração básica

```
1   from logging import basicConfig          # configuração
2   from logging import DEBUG                # Level
3   from logging import debug, info, error   # Cahamadas
4
5   basicConfig(level=DEBUG) # config do level
6
7   # Chamadas do log
8   debug('Mensagem de debug')
9   info('Mensagem informativa')
10  error('Mensagem de erro')
```

A configuração básica

```
1   from logging import basicConf
2   from logging import DEBUG
3   from logging import d      g, info
4
5   basicConfig(level=DEBUG) # config do level
6
7   # Chamadas do log
8   debug('Mensagem de debug')
9   info('Mensagem informativa')
10  error('Mensagem de erro')
```

Vamos trocar os níveis agora

Outras configurações básicas

```
1   from logging import basicConfig
2   from logging import DEBUG
3
4   basicConfig(
5       level=DEBUG,
6       filename='meus_logs.txt',
7       filemode='a',
8       encoding='utf-8',
9       format='%(levelname)s:%(asctime)s:%(message)s'
10  )
```

Mas quando usar cada nível?

Nível	Quando?
DEBUG	Expor eventos necessários para possíveis problemas desenvolvimento ou em ambientes de teste
INFO	Eventos somente informativos. Uma chamada externa, um login, aguma verificação rotineira. Mas somente informativos
WARN	Eventos inesperados, que não resultaram em erros, porém devem ser analisadas com atenção depois
ERROR	Eventos não conseguiram ser executados, alguma coisa não está funcionando como deveria funcionar
CRITICAL	Um evento, ou alguma parte da aplicação está comprometida. Faz com que o sistema não atue da maneira que deveria operar.

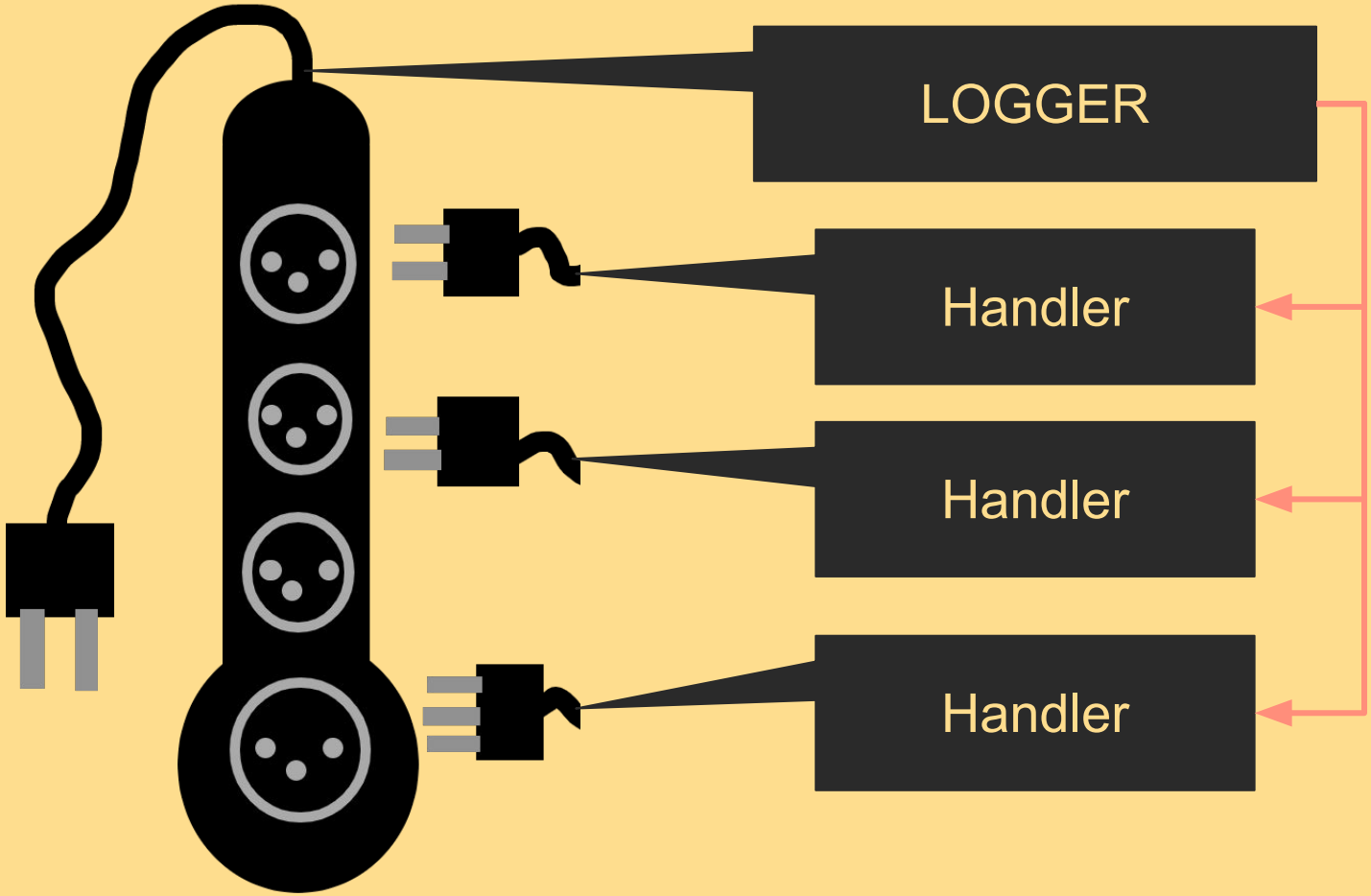
Não tem uma forma de fazer o log escrever no terminal e também escrever nos arquivos?

Um problema!

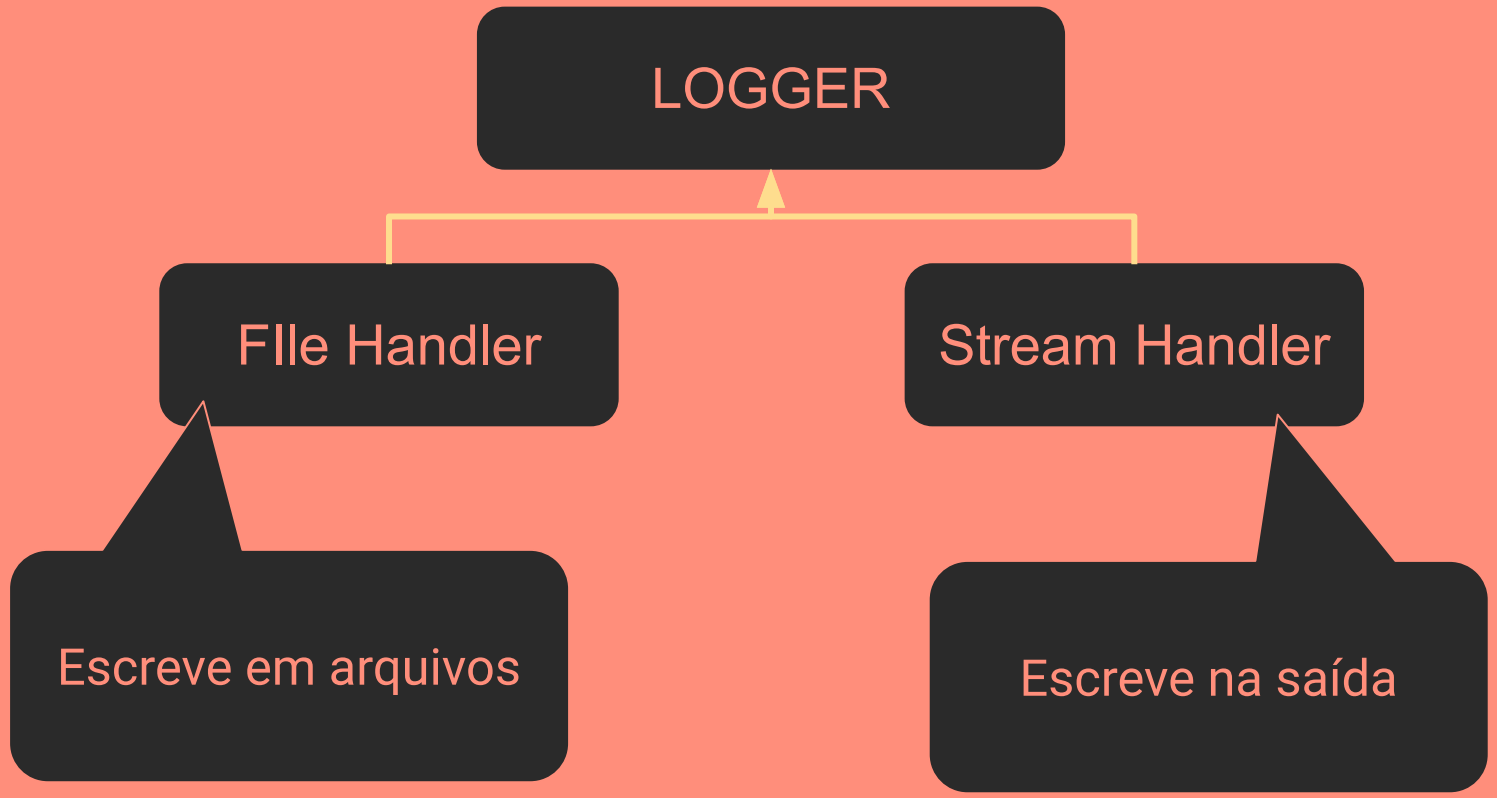
O mecanismo

Que faz tudo funcionar!

As ferramentas do logging



Resolvendo o problema!



O código

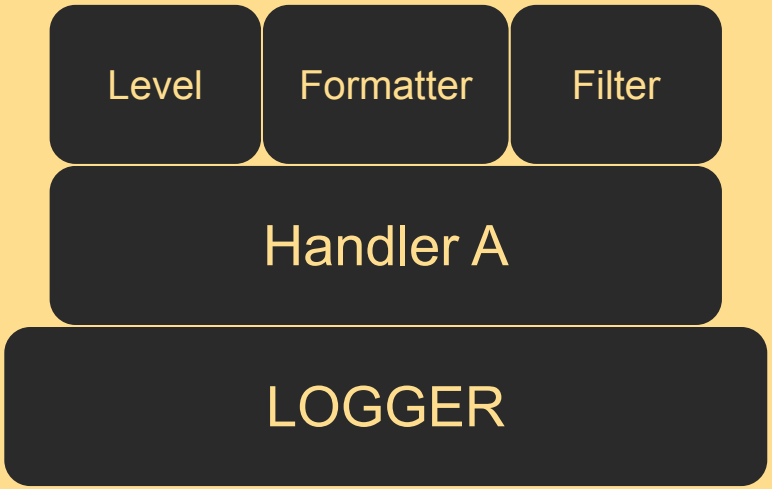
```python
from logging import basicConfig          # configuração
from logging import DEBUG                # Level
from logging import debug, info, error   # Cahamadas
from logging import FileHandler, StreamHandler  # Handlers


basicConfig(
    level=DEBUG,
    encoding='utf-8',
    format='%(levelname)s:%(asctime)s:%(message)s',
    handlers=[FileHandler("meus_logs.txt", "w"), StreamHandler()]
)
```

Variedade de Handlers

- **StreamHandler**
  - Escreve no shell
- **FileHandler**
  - Escreve em um arquivo
- **RotatingFileHandler**
  - Rotaciona o arquivo, quando ficar grande de mais
- **SMTPHandler**
  - Envia um email com a mensagem de log
- **HTTPHandler**
  - Envia um request
- **SysLogHandler**
  - Passa o log para os logs do sistema
- **...**

https://docs.python.org/3/howto/logging.html#useful-handlers

Estrutura dos Handlers




```python
from logging import basicConfig
from logging import DEBUG, INFO
from logging import FileHandler, StreamHandler
from logging import Formatter


formater_file_handler = Formatter(
    '%(asctime)s  %(name)s  %(levelname)s: %(message)s'
)
file_handler = FileHandler("meus_logs.txt", "w")
file_handler.setLevel(INFO)
file_handler.setFormatter(formater_file_handler)


basicConfig(
    level=DEBUG,
    encoding='utf-8',
    format='%(levelname)s:%(asctime)s:%(message)s',
    handlers=[file_handler, StreamHandler()]
)
```

Filtros

A ideia dos filtros é filtrar ou níveis, ou mensagens específicas. Pode ser bom para quando existe a possibilidade de em debug usar dados sensíveis.

```python
from logging import critical, FileHandler, Formatter

class MeuFiltro(Filter):
    def filter(self, record):
        if record.msg.lower().startswith('senha'):
            return False
        return True

file_handler = FileHandler("meus_logs.txt", "w")

file_handler.addFilter(MeuFiltro())

critical('senha 1234')  # Filtra
critical('test filtro') # Não filtra
```

Mais curiosidades sobre Handlers

- Podem ter níveis de logs diferentes
  - Um pode ser DEBUG, enquanto outro pode ser CRITICAL
- Podem ter formatadores diferentes
- Podem ter filtros diferentes
- **Podem ser configurados em arquivos de configuração**

PARA APRENDER MAIS SOBRE ESSE ASSUNTO.

ASSISTA A LIVE #49

loguru

Logs mais simples!

pip install loguru

Loguru

O loguru vem com a ideia de simplificar toda a configuração de logs:

- Sem muita configuração
- Sem Handlers
- Sem filter
- Sem formaters

Mas tudo que aprendemos faz sentido ainda.

Uso básico

```python
from loguru import logger

logger.debug('Debug')
logger.info('Info')
logger.warning('Warning')
logger.error('Error')
logger.critical('Critical')
```

Uso básico

```python
from loguru import logger

logger.debug('Debug')
logger.info('Info')
logger.warning('Warning')
logger.error('Error')
logger.critical('Critical')
```

Tudo formatado e colorido xD

```
(.venv|py-3.10.4 babbage in ~/live_198
o → python loguru_exemplo.py
2022-04-11 16:37:46.163 | DEBUG    | __main__:<module>:3 - Debug
2022-04-11 16:37:46.163 | INFO     | __main__:<module>:4 - Info
2022-04-11 16:37:46.163 | WARNING  | __main__:<module>:5 - Warning
2022-04-11 16:37:46.163 | ERROR    | __main__:<module>:6 - Error
2022-04-11 16:37:46.163 | CRITICAL | __main__:<module>:7 - Critical
```

Um novo "handler"

```python
from sys import stderr
from loguru import logger

logger.remove()

logger.add(
    sink=stderr,
    format='{time} {level} {message} {file}',
    filter=lambda rec: 'senha' not in rec['message'].lower(),
    level='INFO'
)

logger.critical('senha')
logger.debug('Debug')
logger.info('Info')
logger.warning('Warning')
```

Um novo "handler"

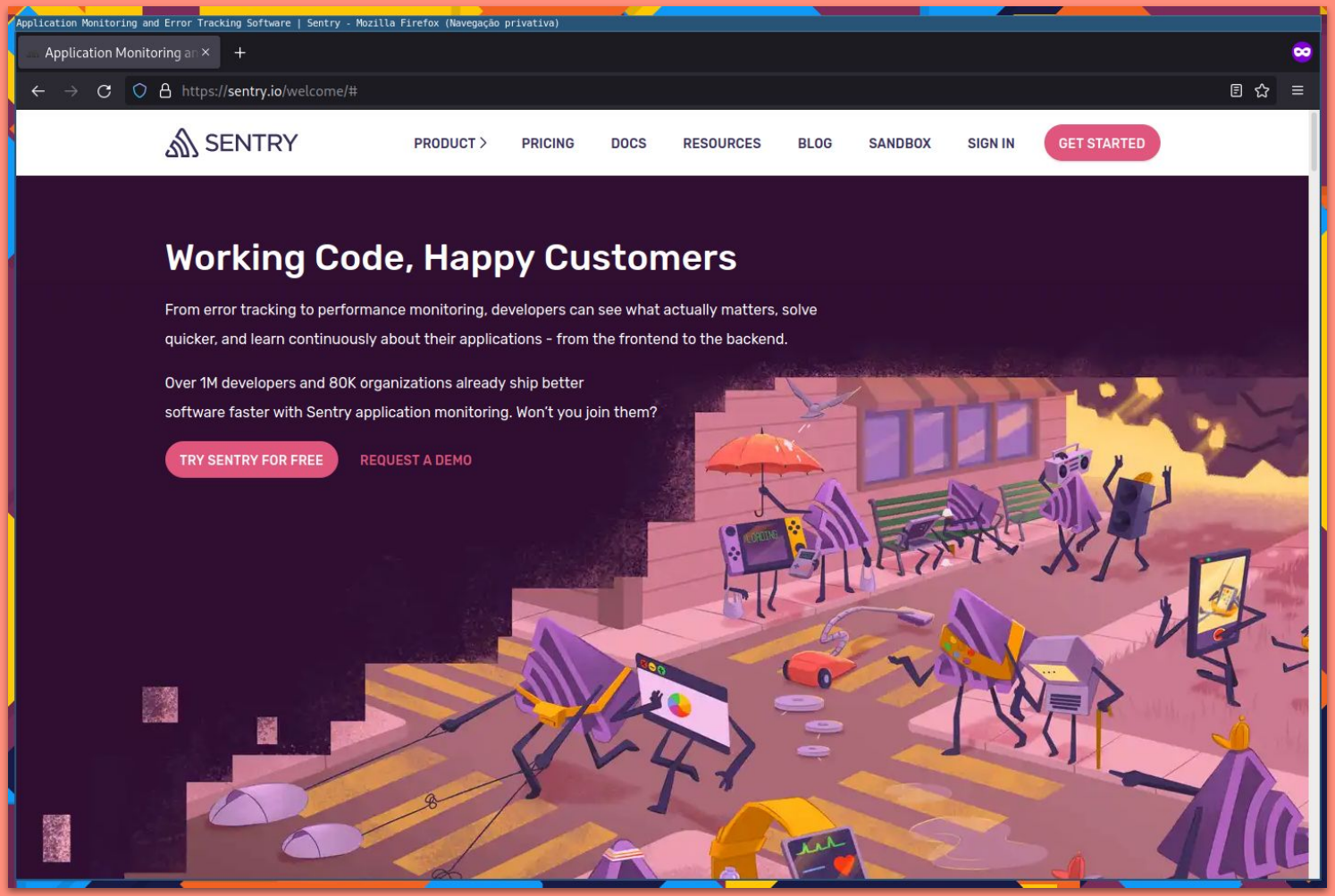
```python
from sys import stderr
from loguru import logger

logger.remove()

logger.add(
    sink=stderr,
    format='{time} {level} {message} {file}',
    filter=lambda rec: 'senha' not in rec['message'].lower(),
    level='INFO'
)

logger.critical('senha')
logger.debug('Debug')
logger.info('Info')
logger.warning('Warning')
```

pode ser um arquivo, um path, PathLib, um File-Like ou um handler padrão do python

Logs em produção

Um rápido overview

Vamos de Sentry? [sentry.io]



Integração no código

```python
import logging
import sentry_sdk
from sentry_sdk.integrations.logging import LoggingIntegration

sentry_logging = LoggingIntegration(
    level=logging.INFO,
    event_level=logging.ERROR
)

sentry_sdk.init(
    dsn='',
    integrations=[sentry_logging]
)
```

Integração no código com loguru

```python
from logging import DEBUG
from loguru import logger
from sentry_sdk import init
from sentry_sdk.integrations.logging import LoggingIntegration, EventHandler

init(
    dsn=''
     integrations=[LoggingIntegration(level=None, event_level=None)]
)  # Inicia o sentry

logger.add(
     EventHandler(level=DEBUG), format="{time} {level} {message}",
)  # Adiciona o sentry ao loguru
```

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3