

Abaixo aos prints,  
ode aos logs!

Renan de Assis



# Quem sou eu?

- (auto descrição)



# Quem sou eu?

- Bacharel em Física



# Quem sou eu?

- Bacharel em Física
- Trabalho com engenharia de software no Serasa



# Quem sou eu?

- Bacharel em Física
- Trabalho com engenharia de software no Serasa
- Gosto de vôlei, jogos de tabuleiro e tenho uma tatuagem do desenho Avatar
- Ajudo na organização de eventos Python Brasil



# AVISOS

# AVISOS

- Contexto de aplicações web
- Minhas considerações do que **eu** aprendi na vida
- Tudo aqui é debatível
- Palestra informativa e pra deixar gostinho de curiosidade



# O que são prints?

— □ ×

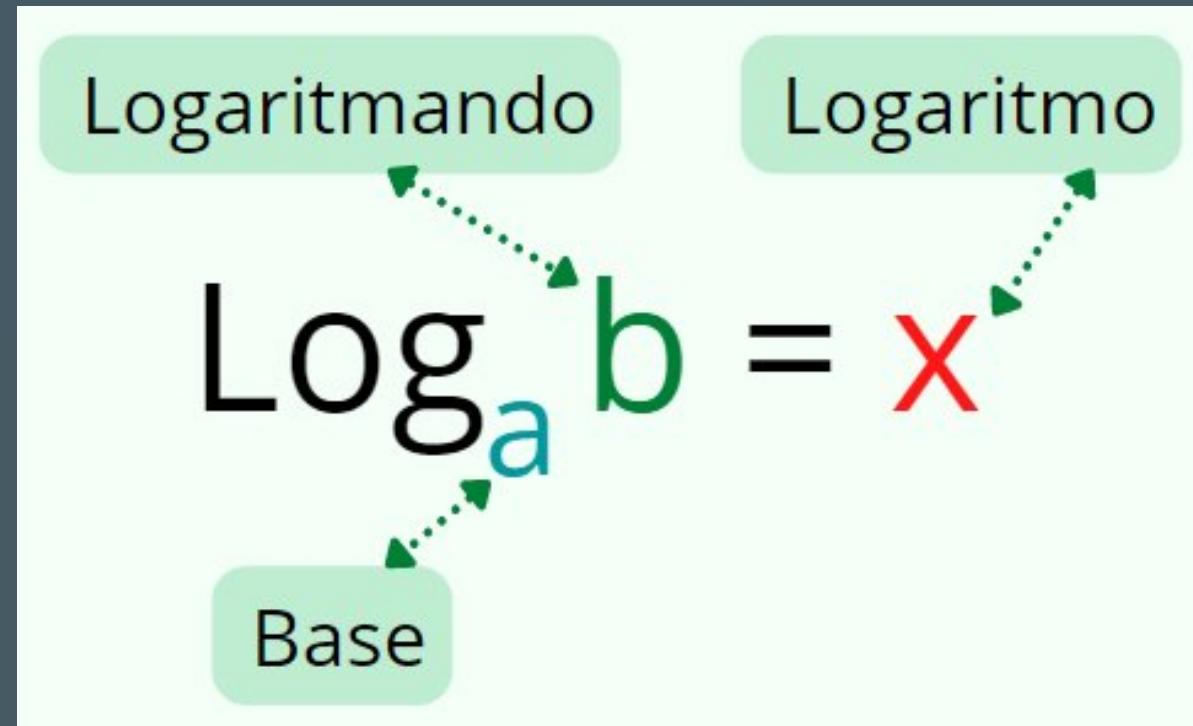
```
1 print("olá mundo")
```

```
2
```

```
3 >>> olá mundo
```

# O que são logs?

# Não tem nada a ver com logaritmo



# O que são logs?

“ “ Expressão utilizada para descrever o processo de **registro** de **eventos** relevantes em um sistema computacional.<sup>1</sup> ” ”

- Registro: "escrever" ou "marcar" **algo** em **algum lugar**
- Eventos que aconteceram no **passado** e podem ser **observados**
- Exemplos de eventos relevantes: **login**, **logoff**, algum erro no sistema, sucesso em uma requisição externa
- Local do registro pode ser a saída do terminal, um arquivo, um servidor de email

1. [https://pt.wikipedia.org/wiki/Log\\_de\\_dados](https://pt.wikipedia.org/wiki/Log_de_dados)

# Quando usar prints?

- Retorno rápido em ambiente local
- Script de uso único

# Quando usar logs?

# Quando usar logs?

## TODO O RESTO

# Logs em python

## [logging](#) — Recurso de utilização do Logging para Python

Código-fonte: [Lib/logging/\\_\\_init\\_\\_.py](#)

---

Este módulo define funções e classes que implementam um registro flexível de eventos de sistema para aplicações e bibliotecas.

O principal benefício de ter a API de registro de eventos a partir de um módulo da biblioteca padrão é que todos os módulos Python podem participar no registro de eventos, de forma que sua aplicação pode incluir suas próprias mensagens, integradas com mensagens de módulos de terceiros.

### Important

Esta página contém informação de referência da API. Para informação tutorial e discussão de tópicos mais avançados, consulte

- [Basic Tutorial](#)
- [Advanced Tutorial](#)
- [Logging Cookbook](#)

# Conceitos de logs - níveis



```
1 import logging  
2  
3 logging.warning("Cuidado! Pouco espaço em disco")  
4 logging.error("Deu algo errado em")  
5 logging.critical("VÃO ADICIONAR LETRAS NOS CPF")  
6  
7 >>> WARNING:root:Cuidado! Pouco espaço em disco  
8 >>> ERROR:root:Deu algo errado em  
9 >>> CRITICAL:root:VÃO ADICIONAR LETRAS NOS CPF
```

# Conceitos de logs - níveis



```
1 import logging  
2  
3 logging.info("Só pra você saber mesmo")  
4  
5 >>>
```

# Conceitos de logs - níveis

| Nível    | Quando é usado   |
|----------|--|
| DEBUG    | Informação detalhada, tipicamente de interesse apenas quando diagnosticando problemas.   |
| INFO     | Confirmação de que as coisas estão funcionando como esperado.  |
| WARNING  | Uma indicação que algo inesperado aconteceu, ou um indicativo que algum problema em um futuro próximo (ex.: 'pouco espaço em disco'). O software está ainda funcionando como esperado. |
| ERROR    | Por conta de um problema mais grave, o software não conseguiu executar alguma função.  |
| CRITICAL | Um erro grave, indicando que o programa pode não conseguir continuar rodando.  |

# Conceitos de logs - formatação



```
1 import logging
2
3 logging.basicConfig(
4     format="%(asctime)s %(levelname)s:%(message)s",
5     datefmt='%d/%m/%Y %I:%M:%S %p'
6 )
7
8 logging.warning("Aviso")
9 logging.error("Erro")
10
11 >>> "06/07/2024 11:08:02 AM WARNING:Aviso"
12 >>> "06/07/2024 11:08:02 AM ERROR:Erro"
```

# Conceitos de logs - armazenamento

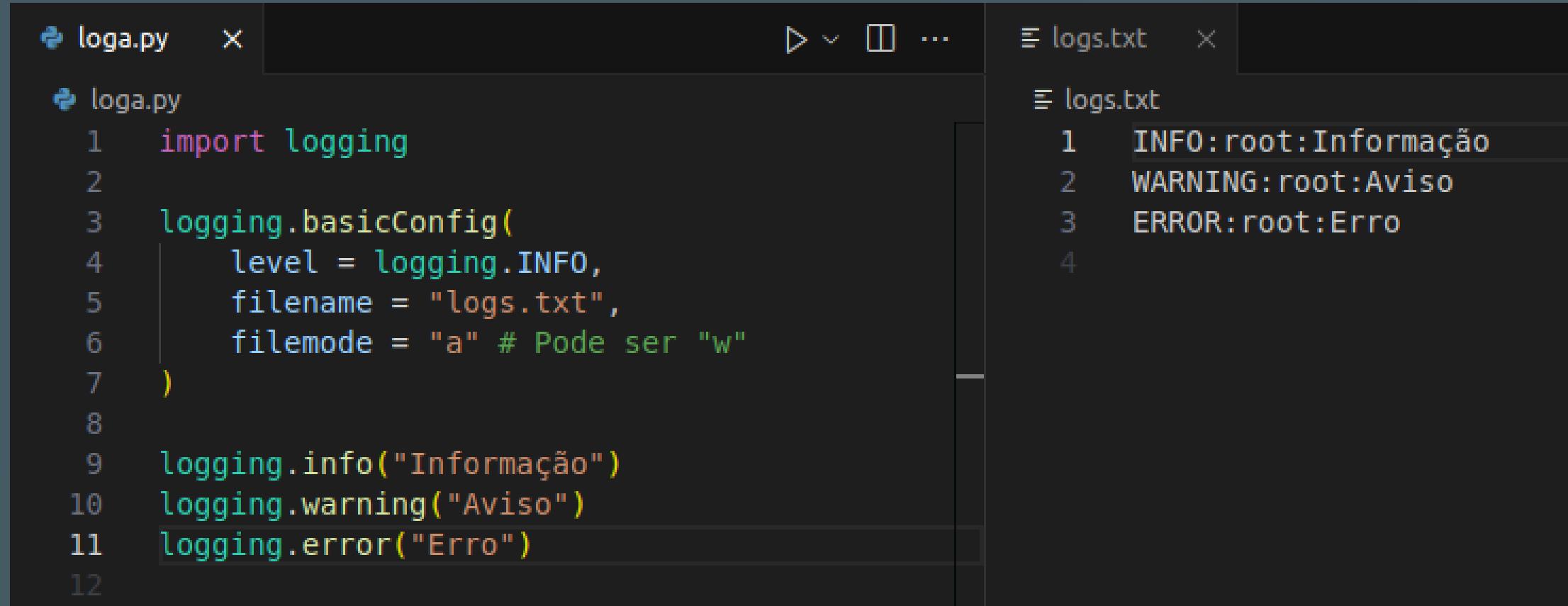
- Relembrando: logs são registros de eventos em algum lugar



```
1 import logging  
2  
3 logging.warning("Tô te avisando que vai dar ruim")  
4  
5 >>> WARNING:root:Tô te avisando que vai dar ruim
```

# Conceitos de logs - armazenamento

- Relembrando: logs são registros de eventos em algum lugar



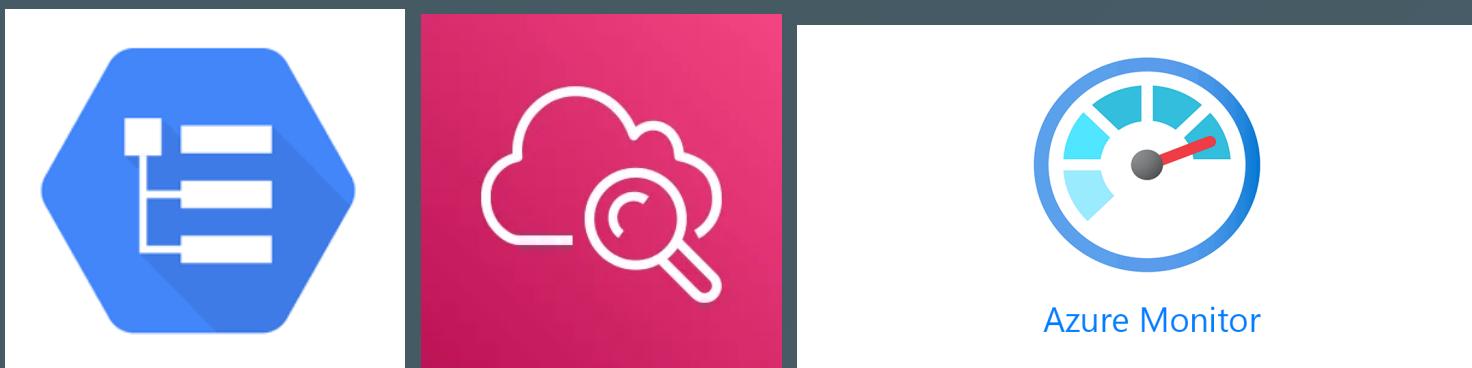
The image shows a code editor interface with two tabs: 'loga.py' and 'logs.txt'. The 'loga.py' tab contains Python code for setting up logging and writing log messages. The 'logs.txt' tab shows the resulting log entries.

```
loga.py
1 import logging
2
3 logging.basicConfig(
4     level = logging.INFO,
5     filename = "logs.txt",
6     filemode = "a" # Pode ser "w"
7 )
8
9 logging.info("Informação")
10 logging.warning("Aviso")
11 logging.error("Erro")
```

```
logs.txt
1 INFO:root:Informação
2 WARNING:root:Aviso
3 ERROR:root:Erro
4
```

# Logs + computação em nuvem

- GCP(Google), AWS(Amazon), Azure(Microsoft), etc
- Serviços para monitorar, armazenar e acessar logs:
  - Cloud Logging (Google)
  - CloudWatch (AWS)
  - Monitor Logs (Azure)



[Query](#)[Recent \(1\)](#)[Saved \(0\)](#)[Suggested \(0\)](#)[Library](#)[Clear query](#)[Save](#)[Stream logs](#)[Run query](#)[Last 1 hour](#) Search all fields[All resources](#)[All log names](#)[All severities](#) Show query

1

1

5

 Log fields Timeline[Create metric](#)[Create alert](#)[Jump to now](#)[More actions](#)[Log fields](#) Search fields and values[RESOURCE TYPE](#)[Google Project](#)[SEVERITY](#)[Notice](#)

2

[Timeline](#)1  
0

4

Jul 4, 9:43PM

10:00 PM

10:30 PM

Jul 4, 10:44 PM

2 results

[Highlight in results](#)[Correlate by](#)[Download](#)

SEVERITY

TIME

BRT

▼

SUMMARY

[Edit](#) Summary fields

Wrap lines

Showing logs for last 1 hour from 7/4/24, 9:43PM to 7/4/24, 10:43PM.

[Extend time by: 1 hour](#)[Edit time](#)

> 2024-07-04 22:15:28.315

cloudroutemanager.googleapis.com CreateProject projects/steel-binder-428501-k3

> 2024-07-04 22:17:09.517

cloudbilling.googleapis.com AssignResourceToBillingAccount projects/steel-binder-428

3

# Logs + ferramentas de observabilidade

- Existem ferramentas muito mais poderosas de monitoramento da performance das aplicações (APM)
- Podem integrar com serviços de computação em nuvem
- Logs, traces, infraestrutura, rastreio de erros, dashboards, etc



# Alternativa à biblioteca logging



# Loguru

Python logging made (stupidly) simple

# Simplificação de logs em python com loguru



```
1 from loguru import logger  
2  
3 logger.info("Ola")  
4 logger.warning("Ola")  
5 logger.error("Ola")  
6 logger.critical("Ola")
```

# Simplificação de logs em python com loguru

```
2024-07-06 19:50:31.631 | INFO      | __main__:<module>:3 - Olá
2024-07-06 19:50:31.632 | WARNING   | __main__:<module>:4 - Olá
2024-07-06 19:50:31.632 | ERROR     | __main__:<module>:5 - Olá
2024-07-06 19:50:31.632 | CRITICAL  | __main__:<module>:6 - Olá
```

# Simplificação de logs em python com loguru

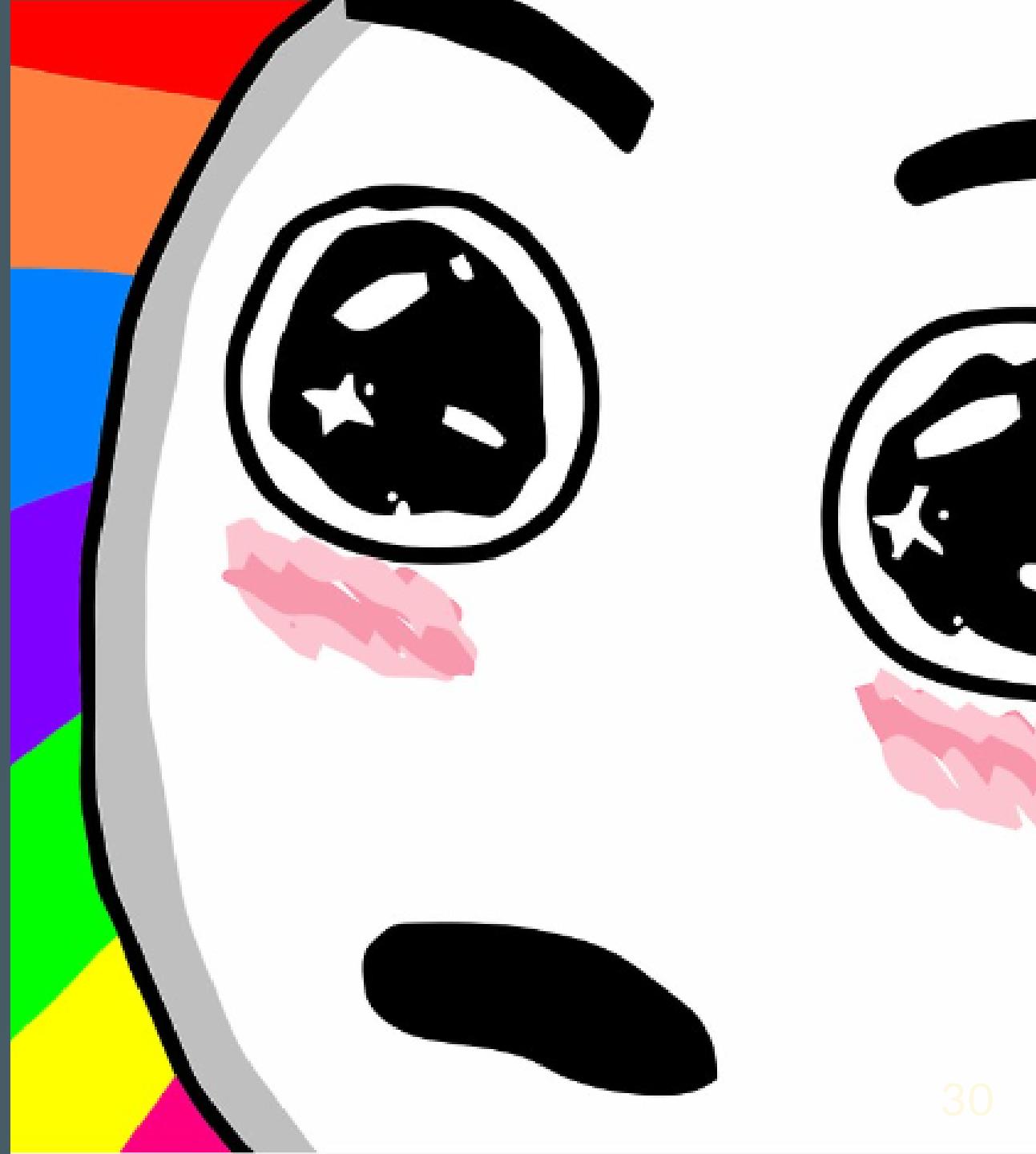
Apenas um comando para configurações

```
1 from loguru import logger  
2  
3 logger.add(  
4     sys.stderr,  
5     format="{time}{level}{message}",  
6     level="INFO"  
7 )
```

# A MELHOR FUNCIONALIDADE

## (na minha opinião)

# Logs estruturados



# Simplificação de logs em python com loguru

```
2024-07-06 22:05:42.968 | INFO      | __main__:<module>:7 - Ola
{"text": "2024-07-06 22:05:42.968 | INFO      | __main__:<module>:7 - Ola\\n", "record": {
    "elapsed": {"repr": "0:00:00.009161", "seconds": 0.009161}, "exception": null, "extra": {}, "file": {"name": "loga.py", "path": "loga.py"}, "function": "<module>", "level": {"icon": "i", "name": "INFO", "no": 20}, "line": 7, "message": "Ola", "module": "loga", "name": "__main__", "process": {"id": 53482, "name": "MainProcess"}, "thread": {"id": 140276529383232, "name": "MainThread"}, "time": {"repr": "2024-07-06 22:05:42.968962-03:00", "timestamp": 1720314342.968962}}}
```

- Saída do log em formato JSON
- Informações facilmente manipuláveis

# Simplificação de logs em python com loguru

```
2024-07-06 22:44:46.704 | INFO      | __main__:<module>:25 - Ola
{"timestamp": 1720316686.704975, "message": "Ola", "severity": "INFO", "log.level": "INFO", "file.name": "loga.py", "line.number": 25, "func.name": "<module>"}
```

# Adicionando contexto

- Consigo adicionar informações personalizadas do meu fluxo

```
1 logger.add(sys.stderr, format="{extra[serialized]})")
2 logger.bind(valor="topper").info("Ola")
```

```
{"timestamp": 1720316767.787056, "message": "Ola", "severity": "INFO", "log.level": "INFO", "file.name": "loga.py", "line.number": 25, "func.name": "<module>", "extra.valor": "topper"}
```

# Adicionando contexto

- E adicionar essas informações em todo um contexto

```
1 with logger.contextualize(usuario="Renan"):  
2     outra_funcao()  
3     logger.info("Acabou o fluxo :( ")
```

– □ ×

```
1 from loguru import logger  
2  
3 def outra_funcao():  
4     logger.info("log de dentro da funcao")
```

# Resumo da ópera

- Logs são registros de eventos relevantes em um sistema computacional
- Usem prints apenas em situações específicas
- A lib de logging padrão é extremamente poderosa e personalizável
- Usem, testem, fuçem o loguru

# Agradecimentos

- Eduardo Mendes (vide Dunossauro) pela [live de python N°198](#) sobre logs que inspirou essa palestra



# Dúvidas?

Linkedin: /in/renan-asantos/

Telegram: @renan\_asantos

Github: renan-asantos

