



Python Brasil 2021

Como capturar exceções de maneira efetiva

@hello@guilatrova.dev

 @guilatrova

 guilatrova

 <https://guilatrova.dev>

Pitch





```
class Service:
    def emit(self, order_id: str) -> dict:

        try:
            ...
        except Exception as e:
            logger.error(...)          # 🖱 Erro 1
            raise e                    # 🖱 Erro 2

        try:                           # 🖱 Dá pra melhorar
            ...
        except Exception as e:
            logger.error(...)
            raise e

        try:                           # 🖱 Dá pra melhorar
            ...
        except Exception as e:
            raise Exception("...")     # 🖱 Dá pra melhorar

        ...
    return x                           # 🖱 Dá pra melhorar
```

Erros comuns

Python é **muito flexível!**

E como diz um famoso tio por aí:

"Com grande ~~poderes~~ flexibilidade, vem grande responsabilidade."

Notei que na maioria dos projetos:

- Usam `logger.error` ao invés de **`logger.exception`**
- Usam `"raise e"` ao invés de só **`"raise"`**
- Não sabem que existe o **`"raise from"`**
- Usam **`descrições muito diferentes`** pras mesmas exceções

ROADMAP

Você está aqui



Regra de negócio

Vamos entender a **necessidade da empresa**, pra podermos discutir o que o código faz.

Refatoração

Vou mostrar um código real utilizado em uma empresa, vamos **analisar** o código, identificar **erros**, e comentar **boas práticas**!

Resultado

Vamos ver **como o código ficou** com as sugestões apresentadas!

Ferramentas

Sugestão de **como lembrar de tudo** isso!



Python Brasil 2021

Regras de negócio com **exemplos reais**

De uma empresa **fictícia** pra evitar ~~processo~~ mal entendido



Python Brasil 2021

Regras de negócio com **exemplos reais**

De uma empresa **fictícia** pra evitar ~~processo~~ mal entendido



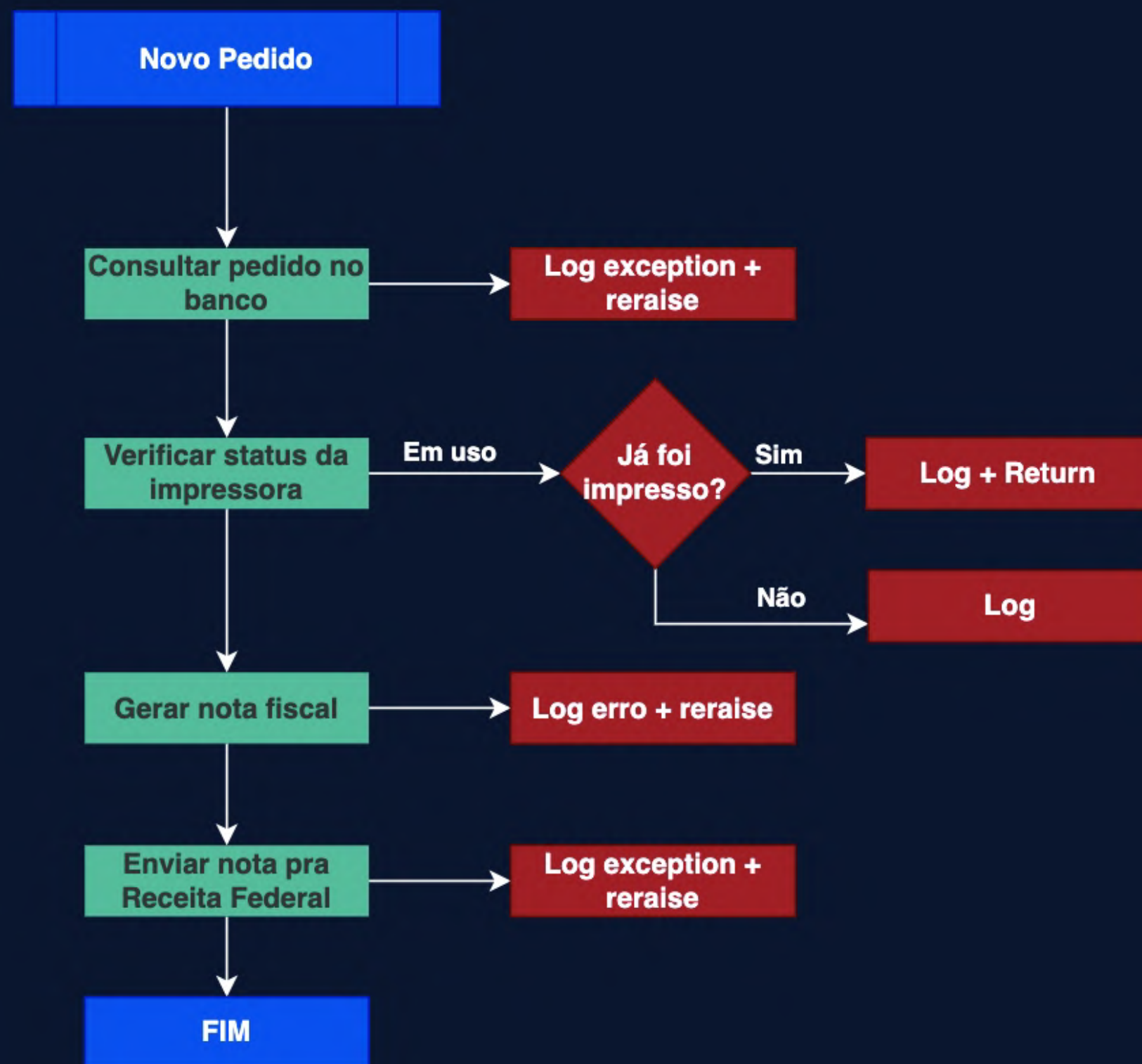


Fluxo

É um micro serviço especializado em **receber pedidos** em uma cozinha (tão fictícia quanto a empresa) e **emitir notas fiscais**.

Os passos são:

1. Pedido novo chega
2. Consultamos o pedido no banco de dados
3. Verificamos o status da impressora
4. Geramos a nota fiscal
5. Emitimos a nota fiscal pra Receita Federal





Python Brasil 2021

Regras de negócio com **exemplos reais**

Como é a função ~~real~~ fictícia

```
class OrderService:
    def emit(self, order_id: str) -> dict:

        try:
            order_status = status_service.get_order_status(order_id)
        except Exception as e:
            logger.exception(
                f"Order {order_id} was not found in db "
                f"to emit. Error: {e}."
            )
            raise e

        (
            is_order_locked_in_emission,
            seconds_in_emission,
        ) = status_service.is_order_locked_in_emission(order_id)
        if is_order_locked_in_emission:
            logger.info(
                "Redoing emission because "
                "it was locked in that state after a long time! "
                f"Time spent in that state: {seconds_in_emission} seconds "
                f"Order: {order_id}, "
                f"order_status: {order_status.value}"
            )

        elif order_status == OrderStatus.EMISSION_IN_PROGRESS:
            logger.info("Aborting emission request because it is already in progress!")
            return {"order_id": order_id, "order_status": order_status.value}

        elif order_status == OrderStatus.EMISSION_SUCCESSFUL:
            logger.info(
                "Aborting emission because it already happened! "
                f"Order: {order_id}, "
                f"order_status: {order_status.value}"
            )
            return {"order_id": order_id, "order_status": order_status.value}
```

```
        try:
            receipt_note = receipt_service.create(order_id)
        except Exception as e:
            logger.exception(
                "Error found during emission! "
                f"Order: {order_id}, "
                f"exception: {e}"
            )
            raise e

        try:
            broker.emit_receipt_note(receipt_note)
        except Exception as e:
            logger.exception(
                "Emission failed! "
                f"Order: {order_id}, "
                f"exception: {e}"
            )
            raise e

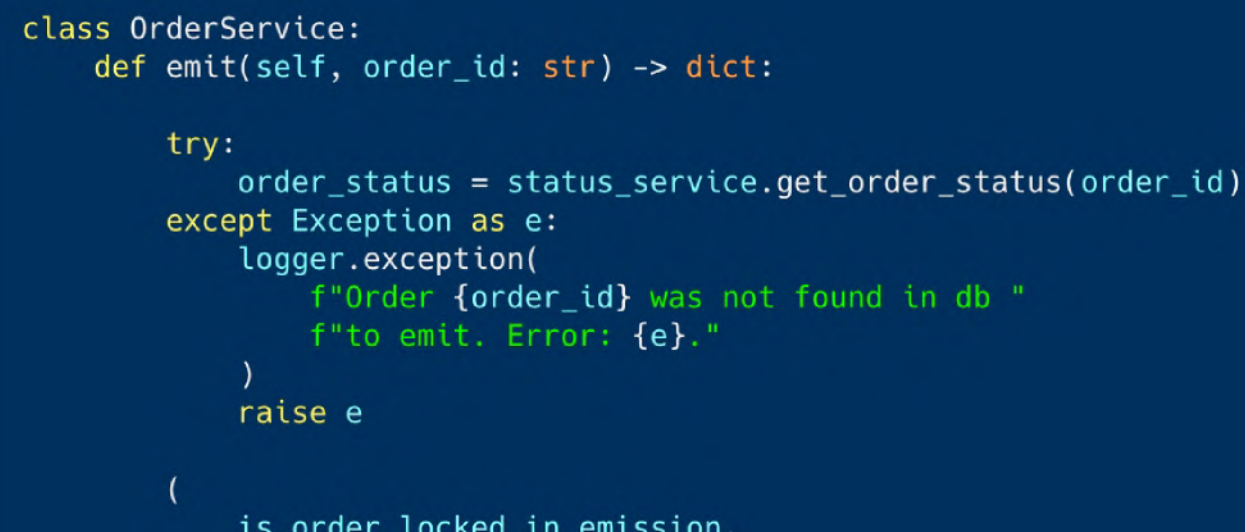
        order_status = status_service.get_order_status(order_id)
        return {"order_id": order_id, "order_status": order_status.value}
```

Isso não é um Service! Isso é um BLOB!

Por quê?

Esse "service" **sabe demais!!**

- Como **produzir erros**
- Como **gerenciar** os erros de outros serviços
- (E é **impossível reutilizar** os serviços sem duplicar código)



```
class OrderService:
    def emit(self, order_id: str) -> dict:

        try:
            order_status = status_service.get_order_status(order_id)
        except Exception as e:
            logger.exception(
                f"Order {order_id} was not found in db "
                f"to emit. Error: {e}."
            )
            raise e

    (
        is order locked in emission
```



Fonte: <https://sourcemaking.com/antipatterns/the-blob>



Criar `exceptions` específicas

```
class OrderCreationException(Exception):
    pass

class OrderNotFound(OrderCreationException):
    def __init__(self, order_id):
        self.order_id = order_id
        super().__init__(
            f"Order {order_id} was not found in db "
            "to emit."
        )

class ReceiptGenerationFailed(OrderCreationException):
    def __init__(self, order_id):
        self.order_id = order_id
        super().__init__(
            "Error found during emission! "
            f"Order: {order_id}"
        )

class ReceiptEmissionFailed(OrderCreationException):
    def __init__(self, order_id):
        self.order_id = order_id
        super().__init__(
            "Emission failed! "
            f"Order: {order_id} "
```



Criar `exceptions` específicas

```
try:
    order_status = status_service.get_order_status(order_id)
except Exception as e:
    logger.exception(...)
    raise OrderNotFound(order_id) from e

...

try:
    ...
except Exception as e:
    logger.exception(...)
    raise ReceiptGenerationFailed(order_id) from e

try:
    broker.emit_receipt_note(receipt_note)
except Exception as e:
    logger.exception(...)
    raise ReceiptEmissionFailed(order_id) from e
```

`raise from`
(exception chaining)

É usado pra identificar a `causa de uma exception`.

Adicionado no PEP 3134.



Python Brasil 2021

Refatorando a função

Mas precisa mesmo usar o **raise from**?



Python Brasil 2021

Refatorando a função

Mas precisa mesmo usar o `raise from`?

“Explicit is **better** than implicit”



Mas precisa mesmo usar o `raise from`?

“Explicit is **better** than implicit”

explicit (com <code>from</code>)	implicit (sem <code>from</code>)
<pre>... raise Exception("I told you it was dangerous") Exception: I told you it was dangerous The above exception was the direct cause of the following exception: Traceback (most recent call last): File ".../exception-chaninig.py", line 24, in <module> ...</pre>	<pre>... raise Exception("I told you it was dangerous") Exception: I told you it was dangerous During handling of the above exception, another exception occurred: Traceback (most recent call last): File ".../exception-chaninig.py", line 24, in <module> ...</pre>

```
...
    raise Exception("I told you it was dangerous")
Exception: I told you it was dangerous

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File ".../exception-chaninig.py", line 24, in <module>
    ...
```

```
...
    raise Exception("I told you it was dangerous")
Exception: I told you it was dangerous

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File ".../exception-chaninig.py", line 24, in <module>
    ...
```



Python Brasil 2021

Refatorando a função

Separar responsabilidades

```
class StatusService:
    def get_order_status(order_id):
        try:
            ...
        except Exception as e:
            raise OrderNotFound(order_id) from e

class ReceiptService:
    def create(order_id):
        try:
            ...
        except Exception as e:
            raise ReceiptGenerationFailed(order_id) from e

class Broker:
    def emit_receipt_note(receipt_note):
        try:
            ...
        except Exception as e:
            raise ReceiptEmissionFailed(order_id) from e
```




Separar responsabilidades

```
class OrderService: # (🐍 Blob)
    def emit(self, order_id: str) -> dict:
        try:
            order_status = status_service.get_order_status(order_id)

            if is_order_locked_in_emission:
                ...

            elif order_status == OrderStatus.EMISSION_IN_PROGRESS:
                ...

            elif order_status == OrderStatus.EMISSION_SUCCESSFUL:
                ...

            receipt_note = receipt_service.create(order_id)
            broker.emit_receipt_note(receipt_note)
            order_status = status_service.get_order_status(order_id)

        except OrderNotFound as e:
            logger.exception(...)
            raise
        except ReceiptGenerationFailed as e:
            logger.exception(...)
            raise
        except ReceiptEmissionFailed as e:
            logger.exception(...)
            raise
        else:
            return {"order_id": order_id, "order_status": order_status.value}
```

(somente) **raise**

É suficiente, e mantém a stacktrace intacta

else

É usado pra destacar "**sucesso**" (nenhuma exception foi gerada).



Python Brasil 2021

Refatorando a função

Sem ser **verboso**

```
class OrderService:
    def emit(self, order_id: str) -> dict:
        try:
            ...
        except OrderNotFound:
            logger.exception("We got a database exception")
            raise
        except ReceiptGenerationFailed:
            logger.exception("We got a problem generating the receipt")
            raise
        except ReceiptEmissionFailed:
            logger.exception("Unable to emit the receipt")
            raise
        else:
            return {"order_id": order_id, "order_status": order_status.value}
```

Sem usar o objeto de **exception**

logger.exception já vai obter a mensagem contida na exception automaticamente.

Use o log pra dar contexto.

Ah, e não precisa mais extrair o objeto 😁.



Python Brasil 2021

Refatorando a função

Sem pedir permissão

```
class OrderService:
    def emit(self, order_id: str) -> dict:
        try:
            order_status = status_service.get_order_status(order_id)

            (
                is_order_locked_in_emission,
                seconds_in_emission,
            ) = status_service.is_order_locked_in_emission(order_id)
            if is_order_locked_in_emission:
                ...

            elif order_status == OrderStatus.EMISSION_IN_PROGRESS:
                ...

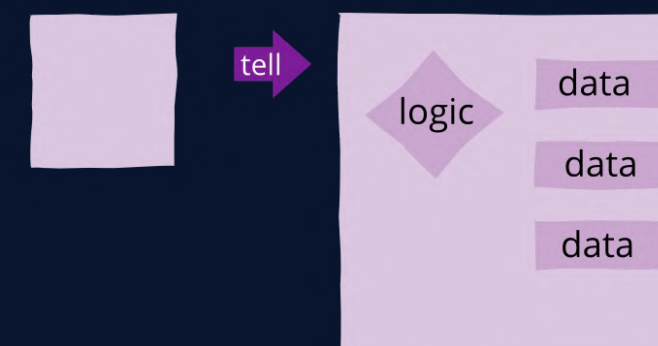
            elif order_status == OrderStatus.EMISSION_SUCCESSFUL:
                ...

            ...
        except:
            ...
```

Princípio: **Tell, Don't Ask**

Não pergunte se pode, **PEÇA!**

Se não for possível, **lance uma exception.**



Fonte: <https://martinfowler.com/bliki/TellDontAsk.html>



Python Brasil 2021

Refatorando a função

Sem pedir permissão

```
class StatusService:
    def ensure_order_unlocked(order_id):
        order_status = ... # query

        if not order_status:
            raise OrderNotFound(order_id)

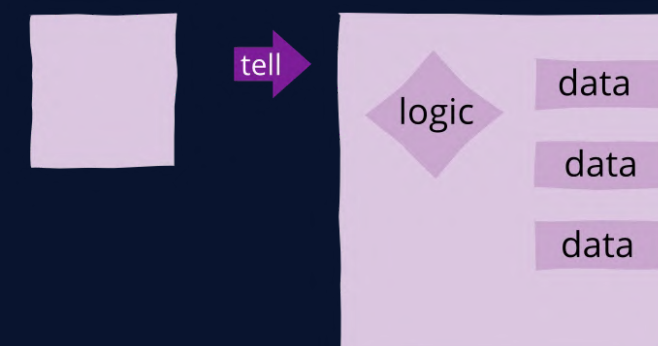
        if order_status == OrderStatus.EMISSION_IN_PROGRESS:
            raise OrderAlreadyInProgress(order_id)

        if order_status == OrderStatus.EMISSION_SUCCESSFUL:
            raise OrderAlreadyEmitted(order_id)
```

Princípio: **Tell, Don't Ask**

Não pergunte se pode, **PEÇA!**

Se não for possível, **lance uma exception.**



Fonte: <https://martinfowler.com/bliki/TellDontAsk.html>



Python Brasil 2021

Como ficou a função ~~real~~ fictícia

```
class OrderFacade:
    def emit(self, order_id: str) -> dict:
        try:
            status_service.ensure_order_unlocked(order_id)
            receipt_note = receipt_service.create(order_id)
            broker.emit_receipt_note(receipt_note)
            order_status = status_service.get_order_status(order_id)

        except OrderAlreadyInProgress as e:
            logger.info("Aborting emission request because it is already in progress!")
            return {"order_id": order_id, "order_status": e.order_status.value}

        except OrderAlreadyEmitted as e:
            logger.info(f"Aborting emission because it already happened! {e}")
            return {"order_id": order_id, "order_status": e.order_status.value}

        except OrderNotFound:
            logger.exception("We got a database exception")
            raise

        except ReceiptGenerationFailed:
            logger.exception("We got a problem generating the receipt")
            raise

        except ReceiptEmissionFailed:
            logger.exception("Unable to emit the receipt")
            raise

        else:
            return {"order_id": order_id, "order_status": order_status.value}
```

“Essa função eu **apresentava** pro meu PO”





Python Brasil 2021

Como lembrar de tudo isso?



Python Brasil 2021

Como lembrar de tudo isso?

Use **linters**!

Tryceratops

guilatrova/
tryceratops



A linter to manage all your python exceptions and try/except blocks (limited only for those who like dinosaurs).

5

Contributors

5

Issues

227

Stars

9

Forks



+



=



<https://github.com/guilatrova/tryceratops>



Python Brasil 2021

Gostou?

Acompanhe meus ~~vacios~~ **aprendizados**



<https://youtube.guilatrova.dev>



@guilatrova



<https://blog.guilatrova.dev>



guilatrova



hello@guilatrova.dev

👉 Feedbacks **negativos e positivos** são bem vindos!



Python Brasil 2021



Obrigado!

Gracias!

(Nenhum dinossauro foi ferido nessa apresentação)

