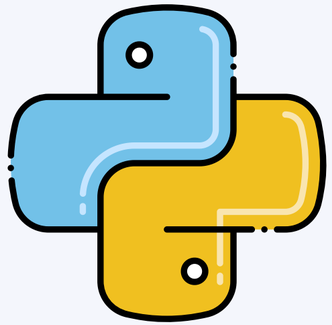
grupy.rn

❯ 21° Meetup GruPy RN 2025

Fazendo Mágica com Python

Emídio Neto
Software Engineer

“No Python Tudo é Objeto



```
3     x = 10
4
5     print(type(x))
6     print(x.bit_length())
7     print(x.__class__)
```

<class 'int'>

4

<class 'int'>

```
print(type(None))
```

```
<class 'NoneType'>
```

```python
def soma(a, b):
    return a + b

f = soma
print(f(2, 3))
print(type(f))

f.foo = "bar"
print(f.foo)
```

```
5
<class 'function'>
bar
```

print(type(type))

<class 'type'>

```
4   import math
5
6   def foo(x):
7       print(x, "->", type(x))
8
9   foo(1)
10  foo("abc")
11  foo(None)
12  foo(math)
13
14  def f(): pass
15  foo(f)
16
17  class C: pass
18  foo(C)
19  foo(C())
```

1 -> **<class 'int'>**

abc -> **<class 'str'>**

None -> **<class 'NoneType'>**

<module 'math' (built-in)> -> **<class 'module'>**

<function f at 0x1010633d0> -> **<class 'function'>**

<class '__main__.C'> -> **<class 'type'> – ??**

<__main__.C object at 0x101038d70> -> **<class '__main__.C'>**

🦆 Duck Typing

→ comportamento importa mais que o tipo

Como tudo é objeto...

Então eu posso alterar o comportamento do que eu quiser em tempo de execução

Monkey Patching

```python
import time

_original_sleep = time.sleep

def patched_sleep(_):
    return None

time.sleep = patched_sleep

time.sleep(10)  # ???
```

O "monkey patching" é uma técnica usada para alterar dinamicamente o comportamento de algum código em tempo de execução.

```python
_real_get = requests.get

def patched_requests_get(*args, **kwargs):
    print(f"Patch: [HTTP] GET {args[0]}")
    return _real_get(*args, **kwargs)

requests.get = patched_requests_get

requests.get("https://example.com")
```

Pra ficar mais fácil entender

Pense no monkey patching como se fosse um decorador não explícito

Mas às vezes pode ser difícil...

* Manter metadados

* Tornar a aplicação de patches reversível e consistente

* Funcionar de forma transparente mesmo se a biblioteca for recarregada ou importada posteriormente

Just use wrapt!

Documentation

https://wrapt.readthedocs.io/en/master/

```python
import wrapt
import requests

def log_get_calls(wrapped, instance, args, kwargs):
    print("[LOG] Chamando requests.get com:", args, kwargs)
    response = wrapped(*args, **kwargs)
    print("[LOG] Status:", response.status_code)
    return response

wrapt.wrap_function_wrapper(requests, "get", log_get_calls)

requests.get("https://example.com")
#[LOG] Chamando requests.get com: ('https://example.com',) {}
#[LOG] Status: 200
```

Um caso de uso da vida real

wrapt example: asgiref toy instrumentation

```python
from wrapt import wrap_function_wrapper
from opentelemetry.instrumentation.utils import unwrap

class AsgirefInstrumentor(BaseInstrumentor):
    def _instrument(self, **kwargs):
        wrap_function_wrapper(asgiref.sync, "async_to_sync", self.__wrapper)

    def _uninstrument(self, **kwargs):
        unwrap(asgiref.sync, "async_to_sync")

    def __wrapper(self, wrapped, instance, args, kwargs):
        with self.tracer.start_as_current_span("async_to_sync", kind=SpanKind.INTERNAL) as span:
            attributes = {
                "exception.type": “async_to_sync”,
                "exception.stacktrace": "\n".join(traceback.format_stack(limit=10))
            }
            span.add_event(name="exception", attributes=attributes)
        return wrapped(*args, **kwargs)
```

você provavelmente usa isso nos testes para simular dependência externa

sim, mas cadê a mágica

```
$ python -c 'print("world")'

> world
```

```
$ python3 -c 'print("world")'
world
```

```
$ PYTHONPATH=.
python -c 'print("world")'
> hello world
```

```
$ PYTHONPATH=.
python -c 'print("world")'
hello world
$ cat sitecustomize.py
print("hello", end=" ")
```

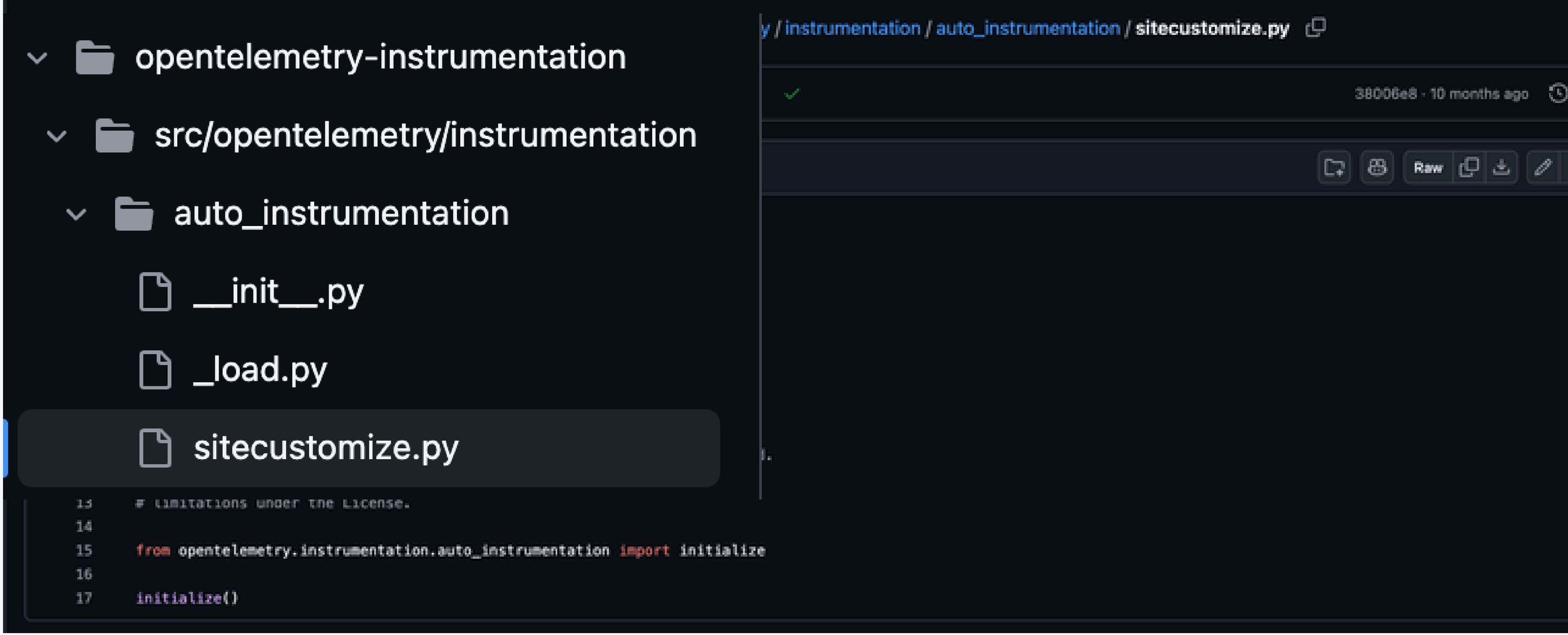
sitecustomize

Por padrão, o módulo site da sua instalação Python tenta carregar um módulo chamado `sitecustomize` do que tiver definido no seu PYTHONPATH.

Referência: https://docs.python.org/3/library/site.html#module-sitecustomize

Juntando Monkey Patching + Sitecustomize

1. **Monkey Patching**
2. **sitecustomize faz a mágica para inicializar os patchings sem mexer no código principal**



auto-instrumentação

Obrigado!

@emdneto