Why you should learn Rust

As a Python developer

Daniel Szoke

About me 👋

About me 👋

🧑‍💻 Software Engineer at Sentry

About me 👋

🧑‍💻 Software Engineer at Sentry

🐍 Python SDK

About me 👋

🧑‍💻 Software Engineer at Sentry

🐍 Python SDK

🦀 Sentry CLI (Rust)

Agenda

1. **Rust vs. Python**
2. Rust's Safety & Reliability Guarantees
3. Rust Tooling
4. Benefits of learning Rust for Python devs

Rust surfaces many **Python** runtime errors at **compile time**

Python

```python
a = None

b = 0

... # a and b are unchanged

print(a >= b) # 🔥 Runtime error!
```

Rust

```rust
let a: Option<i32> = None;
let b = 0;


// ... a and b are unchanged


println!("{}", a >= b); // ❌ compile error
```

Rust

The "fix"

```rust
let a: Option<i32> = None;

let b = 0;


// ... a and b are unchanged


if let Some(a) = a {

    println!("{}", a >= b);

}
```

Rust involves **more upfront effort.** The payoff is **fewer runtime errors.**

Agenda

1. ~~Rust vs. Python~~
2. **Rust's Safety & Reliability Guarantees**
3. Rust Tooling
4. Benefits of learning Rust for Python devs

Explicit mutability
What does this output?

```python
numbers = [0, 1, 2, 3]


mystery(numbers)


print(numbers)
```

Explicit mutability
What does this output?

```python
Python

numbers = [0, 1, 2, 3]


mystery(numbers)


print(numbers)
```

Uncertain

Explicit mutability
What does this output?

```python
Python

numbers = [0, 1, 2, 3]


mystery(numbers)


print(numbers)
```

```rust
Rust

let numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

Uncertain

Explicit mutability
What does this output?

```python
# Python

numbers = [0, 1, 2, 3]


mystery(numbers)


print(numbers)
```

```rust
// Rust

let numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

Uncertain

[0, 1, 2, 3]

Explicit mutability

To mutate, variables must be **declared mutable**!

```rust
let mut numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

Explicit mutability

To mutate, variables must be **declared mutable**!

```rust
let mut numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

`mystery` still cannot mutate here!

Explicit mutability

To mutate, variables must be **declared mutable**!

References also need to be **declared mutable**!

```rust
Rust

let mut numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

`mystery` still cannot mutate here!

Explicit mutability

To mutate, variables must be **declared mutable**!

```rust
                              Rust

let mut numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

mystery still cannot mutate here!

References also need to be **declared mutable**!

```rust
                              Rust

let mut numbers = vec![0, 1, 2, 3];


mystery_mut(&mut numbers);


println!("{:?}", numbers);
```

Ownership and **borrowing rules** ensure **memory safety** at **compile time** without garbage collection

Ownership

```rust
let hello = String::from("Hello, world!");
foo(hello);


println!("{}", hello); // compile error
```

Ownership
Why?

```rust
let hello = String::from("Hello, world!");
foo(hello);


println!("{}", hello); // compile error
```

SENTRY

Ownership

Why?

```rust
let hello = String::from("Hello, world!");
foo(hello); // hello is moved into foo here



println!("{}", hello); // compile error
```

Ownership

Why?

```rust
let hello = String::from("Hello, world!");
foo(hello); // hello is moved into foo here
// foo then drops hello

println!("{}", hello); // compile error
```

Ownership
Fix

```
let hello = String::from("Hello, world!");

foo(hello.clone());


println!("{}", hello);
```

Ownership

Other fix, with borrowing

```rust
let hello = String::from("Hello, world!");
foo2(&hello);


println!("{}", hello);
```

Ownership
Other fix, with borrowing

```
let hello = String::from("Hello, world!");

foo2(&hello);


println!("{}", hello);
```

```
fn foo(s: String) {
    println!("{}", s);
}
```

```
fn foo2(s: &str) {
    println!("{}", s);
}
```

Borrowing
Single borrows

```rust
let mut numbers = vec![0, 1, 2, 3];


mystery(&numbers);


println!("{:?}", numbers);
```

Immutable

```rust
let mut numbers = vec![0, 1, 2, 3];


mystery_mut(&mut numbers);


println!("{:?}", numbers);
```

Mutable

Borrowing
Multiple immutable borrows – ✅

```rust
                            Rust

let numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];


mystery(&numbers);


println!("{:?}", first_two_numbers);
```

Borrowing
Multiple immutable borrows – ✅

```rust
let numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];


mystery(&numbers);


println!("{:?}", first_two_numbers);
```

1st borrow

Borrowing
Multiple immutable borrows – ✅

```rust
let numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];


mystery(&numbers);


println!("{:?}", first_two_numbers);
```

2nd borrow   1st borrow

Borrowing
Mutable & immutable borrow – ❌

```rust
                          Rust

let mut numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];


mystery_mut(&mut numbers);


println!("{:?}", first_two_numbers);
```

Borrowing

Mutable & immutable borrow – ❌

```rust
let mut numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];


mystery_mut(&mut numbers);


println!("{:?}", first_two_numbers);
```
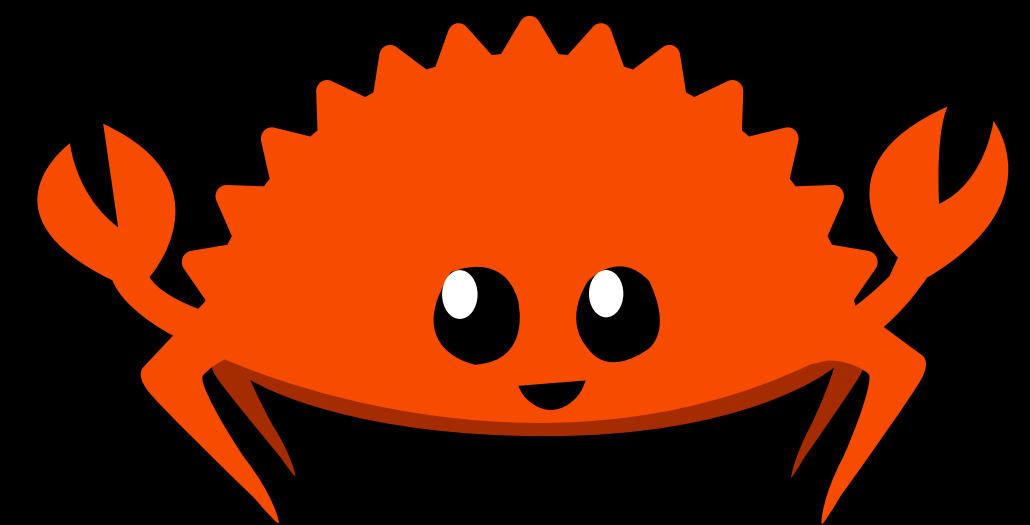
Immutable borrow

Borrowing
Mutable & immutable borrow – ❌

```rust
                          Rust

let mut numbers = vec![0, 1, 2, 3];

let first_two_numbers = &numbers[0..2];


mystery_mut(&mut numbers);


println!("{:?}", first_two_numbers);
```

Compile error! 🔥

Mutable
borrow

Immutable
borrow

SENTRY

Borrowing

Mutable & immutable borrow – ❌

```rust
                        Rust

let mut numbers = vec![0, 1, 2, 3];

let first_two_numbers = &numbers[0..2];


mystery_mut(&mut numbers);


println!("{:?}", first_two_numbers);
```

Compile error! 🔥

Mutable
borrow

Immutable
borrow

* multiple mutable borrows also are not permitted ❌

"Fearless concurrency" – if it compiles, it is thread-safe!

"Fearless concurrency" – if it compiles, it is thread-safe!

Agenda

1. ~~Rust vs. Python~~

2. ~~Rust's Safety & Reliability Guarantees~~

3. **Rust Tooling**

4. Benefits of learning Rust for Python devs

Compiler warnings

```rust
let mut numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];


mystery_mut(&mut numbers);


println!("{:?}", first_two_numbers);
```

Compiler warnings

```rust
Rust

let mut numbers = vec![0, 1, 2, 3];
let first_two_numbers = &numbers[0..2];

mystery_mut(&mut numbers);

println!("{:?}", first_two_numbers);
```

```
error[E0502]: cannot borrow `numbers` as mutable because it is also borrowed as immutable
  --> borrowing/src/main.rs:17:17
   |
15 |     let first_two_numbers = &numbers[0..2];
   |                              ------- immutable borrow occurs here
16 |
17 |     mystery_mut(&mut numbers);
   |                 ^^^^^^^^^^^^ mutable borrow occurs here
18 |
19 |     println!("{:?}", first_two_numbers);
   |                      ---------------- immutable borrow later used here
```

Clippy linter
Standard linter in Rust, with standardized default rules

```rust
pub fn is_positive(number: i32) -> bool {

    if number > 0 {

        return true;

    } else {

        return false;

    }

}
```
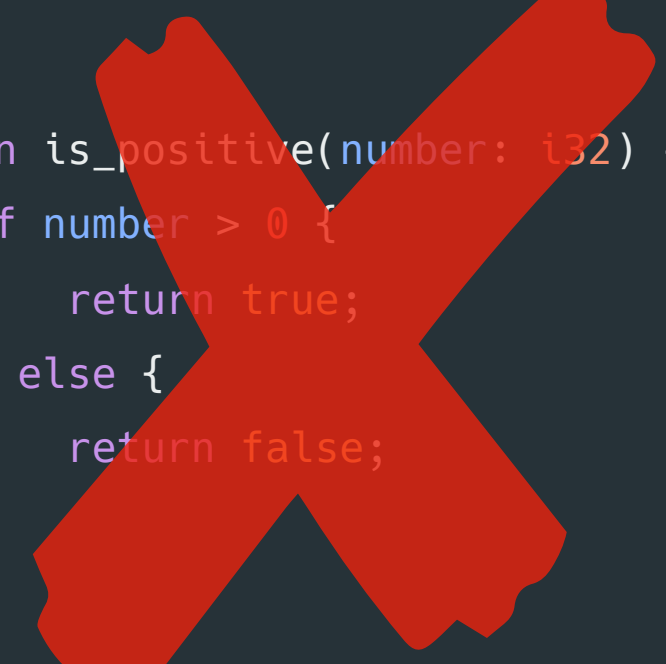
Clippy linter
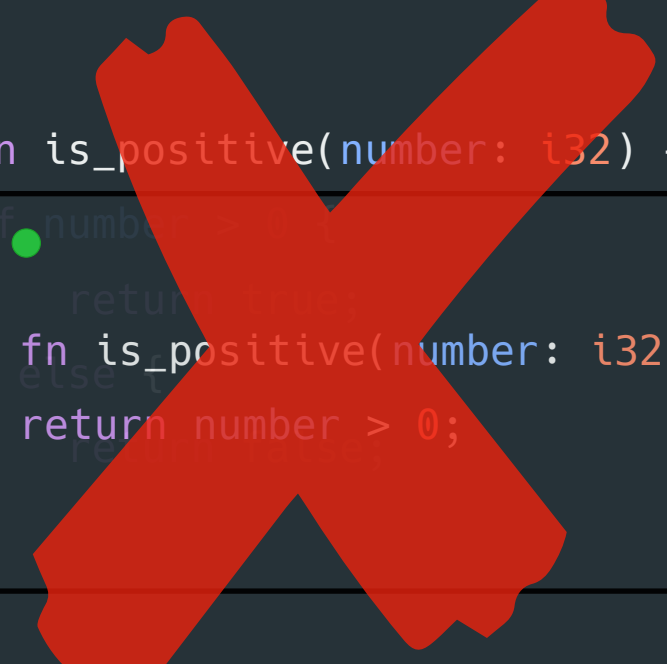**Standard linter in Rust, with standardized default rules**


```
pub fn is_positive(number: i32) -> bool {
    if number > 0 {
        return true;
    } else {
        return false;
    }
}
```

```
pub fn is_positive(number: i32) -> bool {

    return number > 0;

}
```

Clippy linter
Standard linter in Rust, with standardized default rules

```rust
pub fn is_positive(number: i32) -> bool {
    number > 0
}
```

Clippy linter
Standard linter in Rust, with standardized default rules


```rust
pub fn is_positive(number: i32) -> bool {
    number > 0
}
```
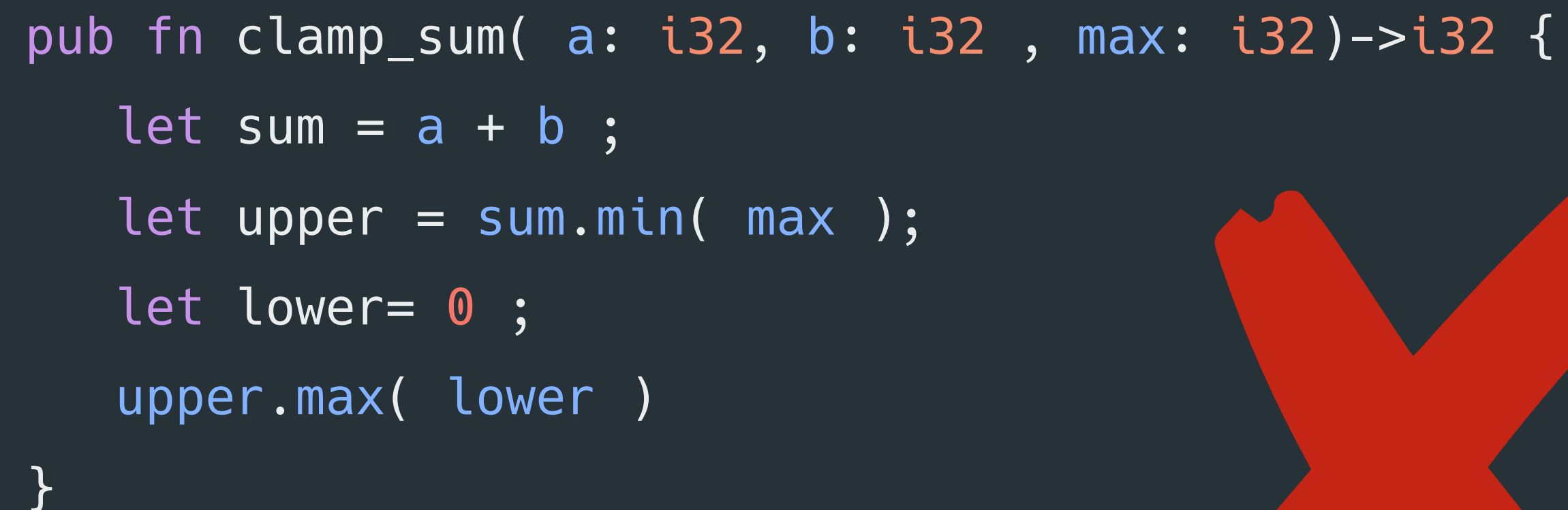
Rustfmt
Standardized formatting for all Rust code

```rust
pub fn clamp_sum( a: i32, b: i32 , max: i32)->i32 {
    let sum = a + b ;

    let upper = sum.min( max );

    let lower= 0 ;

    upper.max( lower )
}
```

Rustfmt
Standardized formatting for all Rust code

```rust
pub fn clamp_sum( a: i32, b: i32 , max: i32)->i32 {
    let sum = a + b ;
    let upper = sum.min( max );
    let lower= 0 ;
    upper.max( lower )
}
```

Rustfmt
Standardized formatting for all Rust code

```rust
pub fn clamp_sum(a: i32, b: i32, max: i32) -> i32 {
    let sum = a + b;

    let upper = sum.min(max);

    let lower = 0;

    upper.max(lower)
}
```

Rustfmt
Standardized — and opinionated — formatting for all Rust code

```rust
pub fn clamp_sum(a: i32, b: i32, max: i32) -> i32 {
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

Rustfmt

Standardized — and opinionated — formatting for all Rust code

```rust
pub fn clamp_sum(a: i32, b: i32, max: i32) -> i32 {
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

```rust
pub fn clamp_sum(a: i32, b: i32, max: i32) -> i32
{
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

```rust
pub fn clamp_sum(
    a: i32,
    b: i32,
    max: i32,
) -> i32 {
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

SENTRY

Rustfmt

Standardized — and opinionated — formatting for all Rust code


```rust
pub fn clamp_sum(a: i32, b: i32, max: i32) -> i32 {
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

```rust
pub fn clamp_sum(a: i32, b: i32, max: i32) -> i32
{
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

```rust
pub fn clamp_sum(
    a: i32,
    b: i32,
    max: i32,
) -> i32 {
    let sum = a + b;
    let upper = sum.min(max);
    let lower = 0;
    upper.max(lower)
}
```

Agenda

1. ~~Rust vs. Python~~

2. ~~Rust's Safety & Reliability Guarantees~~

3. ~~Rust Tooling~~

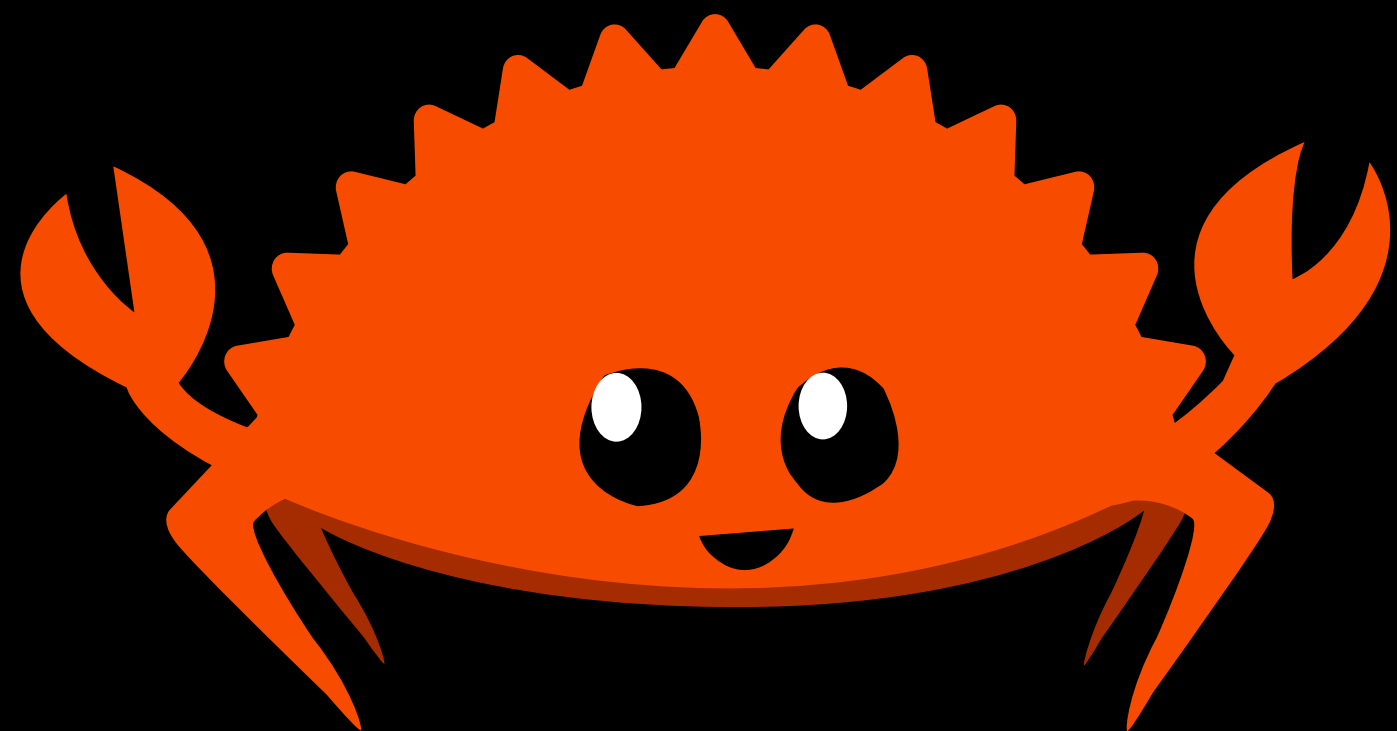4. **Benefits of learning Rust for Python devs**

Rust and Python each have their own advantages — optimize by use case

Rust enforces safety concepts, which are good practice, also in Python

Even if you **never write Rust, learning Rust** will **improve your Python skills**

Ready to learn Rust?

- Check out *The Rust Programming Language*

Sentry is hiring

- My team has an opening for a **Senior Software Engineer** to work on the **Python SDK**

  - Location: Vienna, AT 🇦🇹

- Many other open roles, too!

SENTRY