Introduction to Rust programming language

Rafał Draws

Agenda

- A brief history of Rust
- Typesystem
- Memory Safety Core - Ownership, Borrowing, lifetimes
- Appliences

Who am I

Rafał Draws, MSc. Eng.

- SDE at Nordea Bank
- Rust Poland co-founder
- Music nerd
- Automotive nerd

how can you tell that someone drives a Lexus?

they'll tell you

yes its '99, has 300 thousand miles

What is Rust?

"Rust is a general-purpose programming language. It is noted for its emphasis on performance, type safety, concurrency, and memory safety.

Rust supports multiple programming paradigms. It was influenced by ideas from functional programming, including immutability, higher-order functions, algebraic data types, and pattern matching.

It also supports object-oriented programming via structs, enums, traits, and methods.

Rust is noted for enforcing memory safety (meaning that all references point to valid memory) without a conventional garbage collector;

instead, memory safety errors and data races are prevented by the "borrow checker", which tracks the object lifetime of references at compile time."

~ Wikipedia

But why Rust?

Canadian software developer Graydon Hoare, created Rust in 2006 while working at Mozilla as a side project.

He named the language after a specific type of fungi that is "over-engineered for survival".

A brief history of Rust

Rust interpreter was first written in OCaml, and the language was inspired by programming languages from 1970-1990s, such as: CLU, BETA, Mesa, NIL, Erlang, Newsqueak, Napier, Hermes, Sather, Limbo and Alef.

Hoare described it as "technology from the past come to save the future from itself".

In 2012, in a interview by InfoQ, upon being asked a question: "Why would developers choose Rust", Graydon answered:

- "Our target audience is "frustrated C++ developers". I mean, that's us. So if you are in a similar situation we're in, repeatedly finding yourself forced to pick C++ for systems-level work due to its performance and deployment characteristics, but would like something safer and less painful, we hope we can provide that."

And they did.

Rust history over the years

- 2006, Rust is created in Mozilla Labs
- 2009, Rust is officially incubated by Mozilla
- 2010, The public reveal at Mozilla Summit
- 2012, Rust is rewritten in Rust
- 2014, The "Great Simplification", introducing Ownership/Borrowing instead of Garbage Collection
- 2015, Rust 1.0 is released
- 2018, AWS Firecracker was built to power Lambda and Fargate
- 2020, Discord rewrote Read states from Go to Rust to eliminate Garbage Collection spikes
- 2021, Formation of Rust Foundation
- 2022, Rust merged into Linux Kernel && Cloudflare replaced Nginx with Pingora, proxy written in Rust
- 2023, Windows 11 got oxidized, with core parts of Windows kernel (specifically win32k.sys) rewritten in Rust
- 2024, White House endorses Rust explicitly urging the industry to adopt memory safe languages to secure national infrastructure
- 2025, Rust no longer experimental in Linux Kernel

The History of Rust, by Steve Klabnik


```
echo https://www.youtube.com/watch?v=79PSagCD_AY | qrencode -t utf8i
```

```
snippet +exec is disabled, run with -x to enable
```

How Rust puts emphasis on type safety?

1. Functions must take in typed parameters and return type.
2. Ownership and Borrowing System
3. No nulls (instead Option<T> and Result<T, E>)
4. Immutability by default
5. Exhaustive pattern matching
6. Fearless Concurrency (Send and Sync traits)

How Rust puts emphasis on type safety?

1. Functions must take in typed parameters and return type.
2. Ownership and Borrowing System
3. No nulls (instead Option<T> and Result<T, E>)
4. Immutability by default
5. Exhaustive pattern matching
6. Fearless Concurrency (Send and Sync traits)

Mutability must be defined explicitly with every variable

```
let immutable_string: String = "This is immutable String".to_string();

let mut mutable_string_buffer = "This is mutable".to_string();
mutable_string_buffer.push_str(", and I can modify it!");

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

snippet +exec is disabled, run with -x to enable

And as for 6. Fearless Concurrency - it deserves it's own talk

Typesystem pt.1

Size	Signed Type	Unsigned Type	Description
8-bit	i8	u8	Tiny integers (0 to 255 for unsigned).
16-bit	i16	u16	Small integers. (0 - 2^16)
32-bit	i32	u32	The standard default integer in Rust
64-bit	i64	u64	Large integers (64-bit precision).
128-bit	i128	u128	Massive integers for specialized math.
Arch-dependent	isize	usize	Matches your CPU pointer size (32 or 64-bit).

```
let ptr_size = std::mem::size_of::<usize>();

println!("Pointer size: {} bytes", ptr_size);
println!("Architecture: {}-bit", ptr_size * 8);
```

snippet +exec is disabled, run with -x to enable

Typesystem pt.2

Floating-Points Types

Size	Type	Description
32-bit	f32	32 bit floating point
64-bit	f64	64 bit floating point, default if not specified otherwise

Boolean type

Size	Type	Description
1 byte	true	It's true!
1 byte	false	It's false!

Typesystem pt.3

TUPLE

```
let ip_address: (u8, u8, u8, u8) = (127, 0, 0, 1);
let (a, b, c, d) = ip_address;
println!("Ain't no place like {}.{}.{}.{}", a, b, c, d);

println!("The first octet of an address - {}", ip_address.0);
```

snippet +exec is disabled, run with -x to enable

Typesystem pt.4

Array type

```
let brtdz = [2,1,1,5];

let zip: [u16; 2] = [80,079];

let months = [0; 100];

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

snippet +exec is disabled, run with -x to enable

Typesystem pt.5

Vector

```rust
let first_valid_vec_of_i32: Vec<i32> = vec![-123, i32::MAX, 123, i32::MIN];
let second_valid_vec: Vec<i32> = Vec::new();

let mut stack: Vec<i32> = vec![0, 1, 3];
let popped_from_stack = stack.pop();

println!("Seems like Vec is just like array in python!, {:?} == {:?}", Some(3), popped_from_stack);
```

snippet +exec is disabled, run with -x to enable

char, String, and string slice types

Type	Data Location	Stack Size	Description
String	Heap	24 bytes	Growable, UTF-8 encoded text. Used when you need to modify the data
&str	Heap or Binary	16 bytes	a "slice" or view into the text. The metadata is on stack, but the data is somewhere else
char	Stack	4 bytes	A single unicode scalar value, like 'c'. Always fixed at 4 bytes
[u8; n]	Stack	n bytes	Fixed size array of bytes
Vec<u8>	Heap	24 bytes	A growable array of raw bytes, often used for non-textual data or manual encoding

Functions must take in typed parameters and return type.

This won't compile:

```
fn eat_pizza(pizza) {
  if pizza.source == "piratto" {
    return "why did you do this to me"
  }
}
```

snippet +exec is disabled, run with -x to enable

Functions must take in typed parameters and return type.

This will compile:

```
struct Pizza<'a> {
  source: &'a str
}

fn eat_pizza(pizza: Pizza) {
  if pizza.source == "piratto" {
    println!("mmm yummy")
  } else {
    println!("nice")
  }
}

  let questionmark_pizza: Pizza = Pizza { source: "piratto" };
  eat_pizza(questionmark_pizza);
```

snippet +exec is disabled, run with -x to enable

Ownership and Borrowing

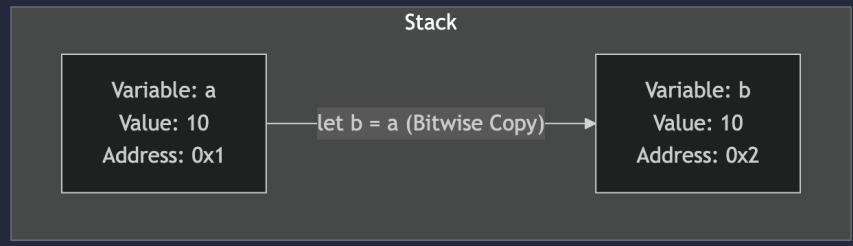
```
let s1 = String::from("Rust"); // s1 owns the memory on the heap

let s2 = s1; // OWNERSHIP IS MOVED HERE from s1 to s2

// This line will cause a COMPILE-TIME ERROR
println!("Value of s1: {}", s1);
```

snippet +exec is disabled, run with -x to enable

- But why all of this hassle?
- Well, this or Garbage Collection
- But it works for some types, right? Like i32, u8, &str (string slice)
- This is due to

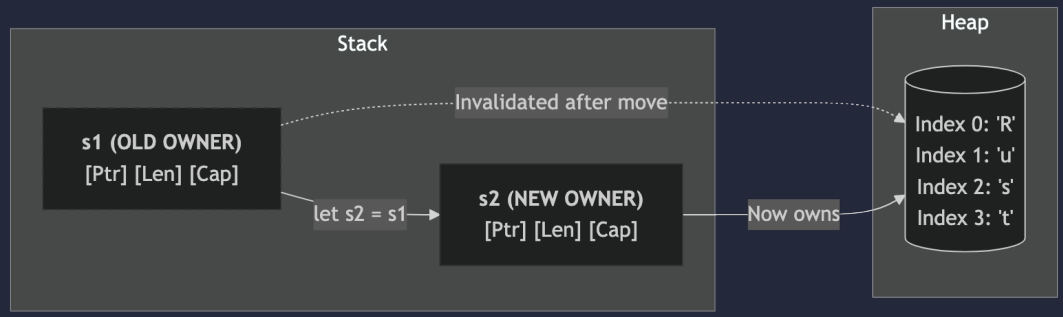
STACK AND HEAP MEMORY ALLOCATION

Stack and Heap memory allocation



For types types of static size, the assignment creates a completely independent twin. Data on stack is fixed size, and duplicating it is faster than managing a pointer.

They implement the `Sized` and `Copy` trait, so on assignment (`let a = b;`), the value is Copied.



This results in invalidation of s1

If both owned the data, they would both try to "drop" (free) that heap memory when the function ends, causing a Double Free crash. By "moving," Rust ensures exactly one variable is responsible for cleaning up.

(Fun fact - NASA dissalows using Heap allocated memory)

Ownership and Borrowing

```rust
fn main() {
    let mut s1 = String::from("hello");

    let r1 = &s1; // Immutable borrow
    let r2 = &s1; // Second immutable borrow (OK!)

    // let r3 = &mut s1; // ERROR: Cannot borrow as mutable while immutable borrows exist.

    println!("{} and {}", r1, r2);
    // r1 and r2 "die" here after their last use.

    let r3 = &mut s1; // OK: No other active borrows!
    r3.push_str(", world");
}
```

snippet +exec is disabled, run with -x to enable

Lifetimes pt.1

Imagine a function that compares two borrowed string slices and returns the longer one.

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

snippet +exec is disabled, run with -x to enable

When you return a reference from a function, Rust needs to guarantee that the reference doesn't point to "garbage" memory.

How to decide which one?

Lifetimes pt.2

We just tell compiler that both lifetimes are valid by the time of return!

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
let first = "I'm learning";
let second = "Rust";

println!("{}!", longest(first, second));
```

snippet +exec is disabled, run with -x to enable

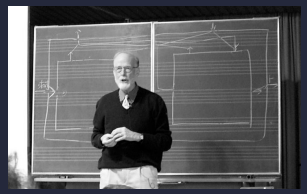
Ownership and Borrowing (last bit I promise)

```
let s1 = String::from("Rust"); // s1 owns the memory on the heap

let s2 = &s1; // IT CREATES A REFERENCE TO THE HEAP ALLOCATED DATA OWNED BY s1

// This line will not cause error anymore
println!("Value of s1: {}", s1);

// and this is valid as well
println!("Value of s2: {}, as well", s2);
```

snippet +exec is disabled, run with -x to enable

No nulls!

▒▒▒ Sir Charles Antony Richard Hoare, a british computer scientist, inventor of Quicksort algorithm and grandfather of "design by contract" famously said:



"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. [...] This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

So how do we omit nulls?

```
Option<T>
Result<Ok, Err>
```

Option of type <T> has two wariants, either Some(T) or None.

For example, if you're creating a pizza structure, you could do it this way:

Let's make a pizza - inital struct

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}
```

Let's make a pizza - throw some meat

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}

struct MeatIngridient(String)
```

Let's make a pizza - source please?

```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}

struct MeatIngridient(String)

enum Pizzeria {
  KUBRYK,
  PIRATTO,
  SOPRANO,
  DOLCEVITA,
  PARMA
}
```

Let's make a pizza - other ingridients

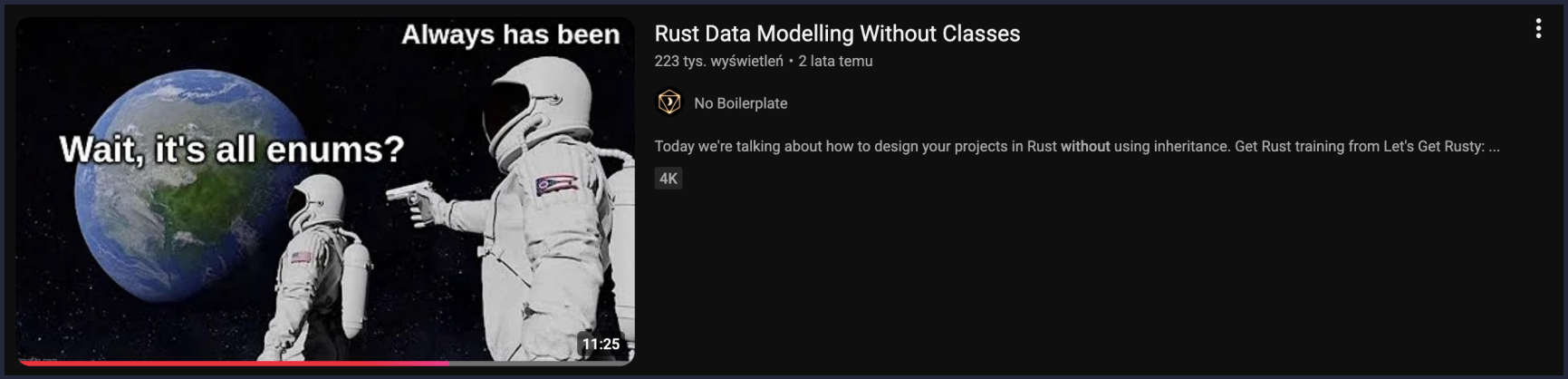
```
struct Pizza {
  source: Pizzeria,
  cheese: String,
  meat: Option<Vec<MeatIngridient>>,
  sauce: Option<Sauce>,
  veggies: Option<Veggie>
}

struct MeatIngridient(String)

enum Pizzeria {
  KUBRYK,
  PIRATTO,
  SOPRANO,
  DOLCEVITA,
  PARMA
}

enum Sauce {
  BIANCO,
  ROSSO,
  WLOCLAWEK,
}

struct Veggie(String)
```

Let's order a pizza

```
let source: Pizzeria = Pizzeria::SOPRANO;
let cheese: String = "Mozzarella".to_string();

let meat: Vec<MeatIngridient> = vec![MeatIngridient("ham".to_string())];

let sauce = Some(Sauce::ROSSO);
let veggies = Some(Veggie("Rocket".to_string()));

let my_wonderful_pizza: Pizza = if sauce.as_ref().is_some_and(|s| *s == Sauce::ROSSO) {
  Pizza {
   source,
   cheese,
   meat: Some(meat),
   sauce,
   veggies
 }
} else {
 Pizza {
   source: Pizzeria::DOLCEVITA,
   cheese,
   meat: Some(vec![MeatIngridient("Ham".to_string()), MeatIngridient("Ham".to_string())]),
   sauce: Some(Sauce::WLOCLAWEK),
   veggies: Some(Veggie("ANANAS".to_string())),
 }
};



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

snippet +exec is disabled, run with -x to enable

More on the type system




```
echo https://www.youtube.com/watch?v=z-0-bbc80JM |
qrencode -t utf8i
```

```
 snippet +exec is disabled, run with -x to enable
```

Appliences

Async:

- Tokio, de-facto standard async runtime and a "language inside rust", a true gem of Rust. Used by AWS, Discord, Cloudflare.
- smol, small, fast, and easy to understand. Great for CLI tooling.
- Embassy, primary choice for async on embedded devices like STM32 and ESP32. Zero-cost async for bare metal

Web & Networking:

- Axum, Web framework, maintained by Tokio team. Uses tower middleware - I would name it FastAPI for Rust (but faster)
- Actix-web, Actor based framework that consistently tops benchmarks
- Hyper, a fast, correct HTTP implementation. Rarely used explicitly, but it almost powers every web library on this list

Databases & Data Handling

- SQLx, my weapon of choice. Pure Rust, async, and compile time checked SQL queries. Supports Postgres, MySQL and SQLite without heavy ORM.
- SeaORM, built on top of SQLx, for Django-like or TypeORM enjoyers
- MongoDB, official, fully async and highly performant mongo driver

Appliences pt.2

Data and ML landscape is shifting from experimental to high-performance alternative.

While python is the choice for experimentation, Rust shines in data intensive infrastructure and production.

Data engineering:

- Polars, so called "pandas killer". Multi threaded by default and significantly faster than pandas. Lazy API for query optimalization (thanks Apache Spark)
- DataFusion, a powerful SQL query engine. Uses Apache Arrow (also written in Rust) to run queries against Parquet, CSV and JSON in lightning speed.
- Delta-rs, a native Rust interface for Delta Lake. Essential for building Lakehouse architectures without JVM/Spark.
- apache-arrow, a backbone of DataFusion. Provides a standardized way to represent columnar data in memory for zero-copy sharing.

Machine Learning:

- burn, PyTorch-like DL framework that can target CPU, GPU (WGPU, CUDA), and even WebAssembly (federated learning!)
- candle, developed by HuggingFace. The "lightweight" ML framework focused on making deployment and serverless inference easy (used heavily for LLMs)
- limfa, the scikit-learn of rust
- tch-rs, a high level wrappers of C++ libtorch. Allows you to run existing PyTorch models in Rust environment (tested it, works wonderful)

Appliences pt.3

WebAssembly and fullstack

- Leptos, a reactive (like SolidJS) full stack framework that feels like React but runs at native speed
- Dioxus, Multi-platform UI. Write once - run everywhere. Uses React like virtual DOM to target Web, Desktop, and Mobile
- Yew, the long standing veteran, also component based, and very stable for enterprise Wasm apps.
- Wasm-bindgen, the bridge between Rust and JS. It handles the "dirty work" of passing strings and objects across the Wasm boundary.

CLI

- Clap, Command line parser. Argparse of Rust.
- Ratatui, Terminal User Interface application, lately enabled to compile bare metal. The future of embedded device development
- Tauri, which uses system's native webview and a Rust backend to create apps that are 10x smaller than discord or slack.

Tools you might not know are written in Rust, but use them in your work

For Python enjoyers

Tool	Replaces	Why
uv	pip, poetry, virtualenv	Install packages 10-100x faster than pip by parallelizing downloads and resolution without Python overhead
Ruff	Flake8	Linter/formatter that processes code on every keystroke
ty	mypy, Pyright, Pylance	Blazingly fast type checker and Language Server (LSP). Uses an incremental architecture to give real-time feedback 10–100x faster than mypy.
polars	pandas	Apache arrow + Rust concurrency
pydantic	pydantic	the core validation logic was moved to pydantic-core, making it 17x faster
cryptography	cryptography	(used by requests, ssh) has migrated its low-level math to Rust for memory safety

For JS enjoyers

Tool	Replaces	Why
SWC	Babel	A compiler that is ~20x faster than Babel, powers Next.js and Deno
Turbopack	webpack	A successor to webpack
biome	prettier + ESLint	single binary that formats and lints JS/TS projects instantly
Tauri	Electron	Already mentioned, but the installers now weight ~3MB instead of ~100MB

For Linux enjoyers

Tool	Replaces	Why
ripgrep (rg)	grep	(rg) Faster searching that respects .gitignore automatically. Built into VS Code.
eza	ls	adds colors, git status dots and icons to file listings
bat	cat	Adds syntax highlighting and git diffs to file output
zoxide (z)	cd	Remembers your most used directories. You type z pro and it jumps to /home/user/projects
Atuin	history	Replaces your shell history with a searchable, syncable SQLite database

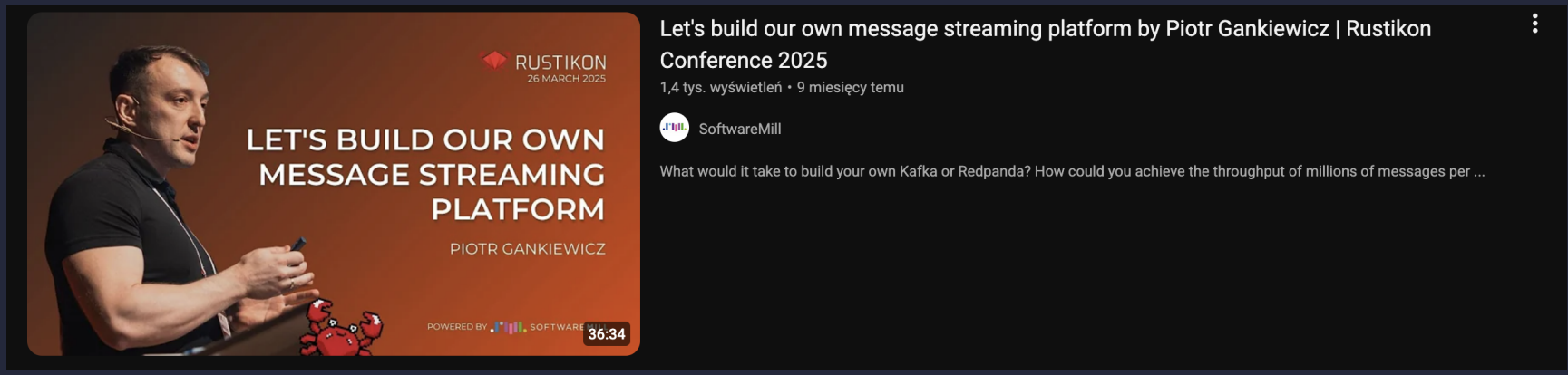
Apache Iggy

I wouldn't be myself if I didn't mention Apache Iggy. Written by Piotr Gankiewicz, is a high-performance, persistent message streaming platform written in Rust.

APACHE
IGGY

I highly suggest familiarizing yourself with this platform, I would name it a Kafka killer.

Apache Iggy




```
echo https://www.youtube.com/watch?v=GkV306PyvqM | qrencode -t utf8i
```

```
snippet +exec is disabled, run with -x to enable
```

Cargo


```
echo https://www.youtube.com/watch?v=wQ_OrmE_AEY | qrencode -t utf8i
```

```
snippet +exec is disabled, run with -x to enable
```

So where to start my Rust journey?

The book

rustup doc --book

or

https://doc.rust-lang.org/book/

Rustlings

https://rustlings.rust-lang.org/

Get inspired

- No Boilerplate
- Code To The Moon
- Let's Get Rusty
- The Rust Programming Language

Allow compiler to be your friend

Few tips

- Go through the book, get bored/excited with theory, do some rustlings
- Read the documentation - the answers are there
- Ensure the LSP has type hightlighting

```rust
let classification: Tensor = model &mut CModule
    .forward_t(xs: &feature_tensor, train: false) Tensor
    .softmax(dim: -1, dtype: Kind::Float);

let mut per_frame_classification: BTreeMap<i64, Vec<f32>> = BTreeMap::new();

for i: i64 in 0..classification.size()[0] {
    let row: Vec<f32> = Vec::<f32>::try_from(classification.get(index: i)).expect("Wrong tensor?");
    per_frame_classification.insert(key: i, value: row);
}

let mut avg_classification: [f32; 5] = [0.0; 5];


for frame: &i64 in per_frame_classification.keys() {
    let vec: &Vec<f32> = per_frame_classification BTreeMap<i64, Vec<f32>>
    .get(key: frame) Option<&Vec<f32>>
    .expect("Should exist");

    vec.iter().enumerate().for_each(|(idx: usize, f: &f32)| avg_classification[idx] += f);
}
```

Experiment

Thank you