perplexity

i mean, usually implementation using syntax something like this

impl xyz for blah {
// code
}

why just
impl<F> Closure<F> {
// code
}

You're pointing out a key distinction in Rust between implementing a *trait* versus providing inherent methods for a *specific type*.

1. Implementing a Trait (`impl Trait for Type`)

The syntax `impl Trait for Type` is used when you want to implement a *trait* for a specific type. A trait is like an interface in other languages; it defines a set of methods that a type must implement to conform to that trait.

Example:

```
trait Summary {
    fn summarize(&self) -> String;
}

struct NewsArticle {
    headline: String,
    location: String,
    author: String,
    content: String,
}

impl Summary for NewsArticle {
    fn summarize(&self) -> String {
        format!("{}, by {} ({})", self.headline, self.author, self.location)
    }
}
```

In this example, `Summary` is a trait, and `NewsArticle` is a struct. The `impl Summary for NewsArticle` block implements the `Summary` trait for the `NewsArticle` type. This means that `NewsArticle` now has a `summarize` method that adheres to the `Summary` trait's definition.

2. Inherent Implementations (`impl Type`)

The syntax `impl Type` (or `impl<Generics> Type` for generic types) is used to define *inherent methods* for a type. These are methods that are directly associated with the type itself, not part of any trait implementation.

Example:

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }

    fn can_hold(&self, other: &Rectangle) -> bool {
        self.width > other.width && self.height > other.height
    }
}
```

Here, `Rectangle` is a struct, and the `impl Rectangle` block defines methods like `area` and `can_hold` that are directly associated with `Rectangle` instances.

Why `impl<F> Closure<F>`?

The `impl<F> Closure<F>` syntax combines these concepts. It means:

1. **Inherent Implementation:** You're adding methods directly to the `Closure` type.
2. **Generic Type:** The `Closure` type is generic, parameterized by the type `F`. The `impl<F>` part declares that this implementation block is also generic over the same type `F`. This allows the methods defined within the `impl` block to work with the generic type `F`.

Therefore, `impl<F> Closure<F>` is *not* implementing a trait. It's defining inherent methods for the `Closure` type, which happens to be a generic type. It's conceptually similar to `impl Rectangle` but for a generic struct. It's providing functionality that's part of the `Closure` type itself, rather than fulfilling the requirements of a separate trait.