Traits and impls

Traits and impls provide a way to define abstract behavior and implement it in a concrete manner. Traits define a set of related functions, while impls provide the actual implementation of those functions. This feature promotes code reusability and modular design.

Example

trait Display { fn display(x: T) -> Array; }

impl DisplayUsize of Display{ fn display(x: usize) -> Array { ... } }

fn main() { // Can be called by the trait name. let a = Display::display(5_usize); // Can be called by the impl name. let b = DisplayUsize::display(5_usize); // Cannot be called by the type name. // T::display(value) - Does not work. } Note that unlike Rust, impls have names, so that they can be explicitly specified.

Impls as generic parameters

In Cairo, impls can be used as generic parameters, allowing for a more flexible and modular design. For example, the following code defines a function that takes a generic parameterT and an implementation of theDisplay trait: fn foo>(value: T) { let a = TDisplay::display(5_usize); let b = Display::display(value); }

Impl inference

When a trait function is called, the compiler will try to infer the impl.

Methods

Impls are used to define methods .

"of" keyword and difference from Rust

In Cairo, theof keyword is used in impls to specify the concrete trait that is being implemented, rather than implementing the trait for a specific type, as is done in Rust with thefor keyword. The main difference between Cairo and Rust in this context is that Cairo doesn’t have a direct type-to-trait implementation relationship. Instead, Cairo emphasizes implementing the trait directly, with the concrete trait name specified after the of keyword.

4.8 Constant items ð§ 5.1 Paths