# Operator Overloading

Operator overloading is a feature in some programming languages that allows the redefinition of standard operators, such as addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`), to work with user-defined types. This can make the syntax of the code more intuitive, by enabling operations on user-defined types to be expressed in the same way as operations on primitive types.

In Cairo, operator overloading is achieved through the implementation of specific traits. Each operator has an associated trait, and overloading that operator involves providing an implementation of that trait for a custom type.

However, it's essential to use operator overloading judiciously. Misuse can lead to confusion, making the code more difficult to maintain, for example when there is no semantic meaning to the operator being overloaded.

Consider an example where two `Potions` need to be combined. `Potions` have two data fields, mana and health. Combining two `Potions` should add their respective fields.

```cairo
{{#include ../listings/ch11-advanced-features/no_listing_01_potions/src/lib.cairo}}
```

In the code above, we're implementing the `Add` trait for the `Potion` type. The add function takes two arguments: `lhs` and `rhs` (left and right-hand side). The function body returns a new `Potion` instance, its field values being a combination of `lhs` and `rhs`.

As illustrated in the example, overloading an operator requires specification of the concrete type being overloaded. The overloaded generic trait is `Add<T>`, and we define a concrete implementation for the type `Potion` with `Add<Potion>`.

{{#quiz ../quizzes/ch11-03-operator-overloading.toml}}