Enums

An enum is a type that can hold one of the types from the list of its defined variants. Each variant has its own type, and when such an enum variable holds a variant, it contains a value of the type of the variant. It is one of the ways to define custom user types. The variants of an enum can be of any defined type, including user-defined types, like other enums orstructs .

Definition

An enum is defined by theenum keyword. Here are a few examples of enum definitions: * An enum with only unit-type * variants: * rust * enum Color { * red: (), * green: (), * blue: (), * } * * An enum with only core type variants: *rust * enum Status { * /// Time passed since submission. * Pending: u32, * /// true for success, false for failure. * Done: bool, * } * * An enum with members of other user-defined types: * rust * enum Car { * /// color * Sedan: Color, * /// color, number of doors * Hatchback: (Color, u32), * } *

Usage

An enum is instantiated with theEnumName::VariantName(value) syntax, wherevalue is of the type of the variantVariantName . For example: let red = Color::red(()); let success = Status::Done(true); let red_sedan = Car::Sedan(Color::red(())); let blue_hatchback = Car::Hatchback((Color::blue(()), 5)); To inspect an enum variable, it can be matched using thematch keyword. A match expression must have an arm for each variant of the enum, that specifies the logic that should happen in the case the enum variable holds that variant. For example: fn car_color(car: Car) -> Color { match car { Car::Sedan(color) => color, Car::Hatchback((color, _)) => color, } } For more details about match expressions, seehere .

4.6 Structs 4.8 Constant items ð§