Modules, Types & Structs

When writing smart contracts you will leverage common programming concepts such:

- Modules
- Data types & Collections
- Classes & Structures

Modules

Modules help you to organize your code and reuse third-party libraries.

The main module you will use in your contract will be theNEAR SDK , which: gives you access to theexecution environment , allows you tocall other contracts ,transfer tokens , and much more.

- JavaScript
- Rust

contract-ts/src/contract.ts loading ... See full example on GitHubcontract-rs/src/lib.rs loading ... See full example on GitHub Using external libraries As a general rule of thumb for Rust, anything that supportswasm32-unknown-unknown will be compatible with your smart contract. However, we do have a size limit for a compiled contract binary which is ~4.19 MB, so it is possible that certain large libraries will not be compatible.

Native Types

When writing contracts you have access toall the language'snative types .

- JavaScript
- Rust

number , bigint ,

string ,

[ ] ,

{ }

... u8 ,

u16 ,

u32 ,

u64 ,

u128 ,

i8 ,

i16 ,

i32 ,

i64 ,

i128 ,

Vec < T

,

HashMap < K , V

... tip Always prefernative types in the contract'sinterface . The only exception is values larger than52 bits (such asu64 andu128), for which string-like alternatives are preferred. warning Always make sure to check forunderflow andoverflow errors. For Rust, simply addoverflow-checks=true in yourCargo .

SDK Collections

Besides the native types, the NEAR SDK implementscollections such asVector andUnorderedMap to help you store complex data in the contract's state.

- JavaScript
- Rust

storage-js/src/index.ts loading ... See full example on GitHub storage-rs/contract/src/lib.rs loading ... See full example on GitHub tip Always preferSDK collections over native ones in the contract'sattributes (state) .

Internal Structures

You can define and instantiate complex objects through classes and structures.

- JavaScript
- Rust

contract-ts/src/model.ts loading ... See full example on GitHub contract-rs/src/donation.rs loading ... See full example on GitHub Notice that the struct is decorated with multiple macros:

- BorshDeserialize
- &BorshSerialize
- allow the structure to be read and
- written into the contract's state
- Serialize
- &Deserialize
- allow the structure to be used as an input type and
- return type of the contract's methods.

tip If you are curious on why the (de)serialization is needed read ourserialization documentation Edit this page Last updatedonApr 24, 2024 byUtku Enes GÜRSEL Was this page helpful? Yes No Need some help?Chat with us or check ourDev Resources ! Twitter Telegram Discord Zulip

Previous Quickstart ✨ Next The Contract Class