Functions

Functions are a fundamental part of any programming language, including Stylus, enabling you to encapsulate logic into reusable components.

This guide covers the syntax and usage of functions, including internal and external functions, and how to return multiple values.

Learn More

- Rust docs - Functions
- Solidity docs - Functions

Overview

A function in Stylus consists of a name, a set of parameters, an optional return type, and a body.

Just as with storage, Stylus methods are Solidity ABI equivalent. This means that contracts written in different programming languages are fully interoperable.

Functions are declared with thefn keyword. Parameters allow the function to accept inputs, and the return type specifies the output of the function. If no return type is specified, the function returnsvoid .

Following is an example of a functionadd that takes twouint256 values and returns their sum.

note This code has yet to be audited. Please use at your own risk. fn

add (a : uint256 , b : uint256)

-> uint256 { return a + b ; }

Function Parameters

Function parameters are the inputs to a function. They are specified as a list ofIDENTIFIER: Type pairs, separated by commas.

In this example, the functionadd_numbers takes twou32 parameters,a andb and returns the sum of the two numbers.

fn

add_numbers (a :

u32 , b :

u32)

->

u32

{ a + b }

Return Types

Return types in functions are an essential part of defining the behavior and expected outcomes of your smart contract methods.

Here, we explain the syntax and usage of return types in Stylus with general examples.

Basic Syntax

A function with a return type in Stylus follows this basic structure. The return type is specified after the-> arrow. Values are returned using thereturn keyword or implicitly as the last expression of the function. In Rust and Stylus, the last expression in a function is implicitly returned, so thereturn keyword is often omitted.

pub

fn

function_name (& self)

->

ReturnType

{ // Function body }

Examples

Function returning a String: Thisget_greeting function returns aString . The return type is specified asString after the-> arrow.

pub

fn

get_greeting ( )

->

String

{ "Hello, Stylus!" . into ( ) } Function returning an Integer: Thisget_number function returns an unsigned 32-bit integer (u32).

pub

fn

get_number ( )

->

u32

{ 42 } Function returning a Result withOk andErr variants: Theperform_operation function returns aResult TheResult type is used for functions that can return either a success value (Ok) or an error (Err). In this case, it returnsOk(value) on success and an error variant ofCustomError on failure.

pub

enum

CustomError

{ ErrorVariant , }

pub

fn

perform_operation (value :

u32)

->

Result < u32 ,

CustomError

{ if value

0

{ Ok (value) }

else

{ Err (CustomError :: ErrorVariant) } }

Public Functions

Public functions are those that can be called by other contracts.

To define a public function in a Stylus contract, you use the#[public] macro. This macro ensures that the function is accessible from outside the contract.

Previously, all public methods were required to return aResult type withVecas the error type. This is now optional. Specifically, if a method is "infallible" (i.e., it cannot produce an error), it does not need to return a Result type. Here's what this means:

- Infallible methods: Methods that are guaranteed not to fail (no errors possible) do not need to use theResult
- type. They can return their result directly without wrapping it inResult
- .
- Optional error handling: TheResult
- type withVec
- as the error type is now optional for methods that cannot produce an error.

In the following example,owner is a public function that returns the contract owner's address. Since this function is infallible (i.e., it cannot produce an error), it does not need to return aResult type.

[external]

impl

Contract

{ // Define an external function to get the owner of the contract pub

fn

owner (& self)

->

Address

{ self . owner . get ( ) } }

Internal Functions

Internal functions are those that can only be called within the contract itself. These functions are not exposed to external calls.

To define an internal function, you simply include it within your contract's implementation without the#[public] macro.

The choice between public and internal functions depends on the desired level of accessibility and interaction within and across contracts.

In the followinge example,set_owner is an internal function that sets a new owner for the contract. It is only callable within the contract itself.

impl

Contract

{ // Define an internal function to set a new owner pub

fn

set_owner (& mut

self , new_owner :

Address)

{ self . owner . set (new_owner) ; } } To mix public and internal functions within the same contract, you should use two separateimpl blocks with the same contract name. Public functions are defined within animpl block annotated with the# [public] attribute, signifying that these functions are part of the contract's public interface and can be invoked from outside the contract. In contrast, internal functions are placed within a separateimpl block that does not have the#[public] attribute, making them internal to the contract and inaccessible to external entities.

src/lib.rs

// Only run this as a WASM if the export-abi feature is not set.

![cfg_attr(not(any(feature =

"export-abi" , test)), no_main)] extern

crate

alloc ;

use

alloc :: vec ; use

stylus_sdk :: alloy_primitives :: Address ; use

stylus_sdk :: prelude :: * ; use

stylus_sdk :: storage :: StorageAddress ;

use

stylus_sdk :: alloy_primitives :: U256 ; use

stylus_sdk :: storage :: StorageU256 ; use

stylus_sdk :: console ;

[storage]

[entrypoint]

pub

struct

ExampleContract

{ owner :

StorageAddress , data :

StorageU256 , }

[public]

impl

ExampleContract

{ // External function to set the data pub

fn

set_data (& mut

self , value :

U256)

{ self . data . set (value) ; }

// External function to get the data pub

fn

get_data (& self)

->

U256

{ self . data . get ( ) }

// External function to get the contract owner pub

fn

get_owner (& self)

->

Address

{ self . owner . get ( ) } }

impl

ExampleContract

{ // Internal function to set a new owner pub

fn

set_owner (& mut

self , new_owner :

Address)

{ self . owner . set (new_owner) ; }

// Internal function to log data pub

fn

log_data (& self)

{ let _data =

self . data . get ( ) ; console! ("Current data is: {:?}" , _data) ; } }

Cargo.toml

[package] name

=

"stylus-functions" version

=

"0.1.0" edition

=

"2021"

[dependencies] alloy-primitives

=

"=0.7.6" alloy-sol-types

=

"=0.7.6" mini-alloc

=

"0.4.2" stylus-sdk

=

"0.6.0" hex

=

"0.4.3" sha3

=

"0.10.8"

[features] export-abi

=

["stylus-sdk/export-abi"] debug

=

["stylus-sdk/debug"]

[lib] crate-type

=

["lib" ,

"cdylib"]

[profile.release] codegen-units

=

1 strip

=

true lto

=

true panic

=

"abort" opt-level

=

"s" Edit this page Previous Constants Next Errors