

Functions

A function is a unit of code that performs some logic. It is defined using the `fn` keyword. Examples of functions are: `fn main() { let x = 3; }`

`fn inc(x: u32) -> u32 { x + 1 }` A function consists of 2 main parts: the function signature and the function body. For example, `// Function signature | | Body starts here // V V fn inc_n(x: T) -> T { x + N }`

Function signature

The function signature defines the function name, the [generic parameters](#), the parameters and the return type. `fn <> [->]` In the example above, the signature is `fn inc_n(x: T) -> T`.

Function name

The function name is the name used to refer to the function. The Cairo convention is to name functions using the 'snake_case' form. In the example above, the function name is `inc_n`.

Parameters

Parameters define the types of the values that are passed to the function when it is called and define [variables](#) to the function's body. Each such variable can be referred to using the parameter name to be used in the function body. The parameters are defined using a comma-separated list of `[] : items`, enclosed by parentheses `((...))`. In the `inc_n` example above, there is one parameter named `x` of the generic type `T`. You can specify modifiers of the function parameters. These can be either `mut` or `ref` (not both). A parameter that is defined with the `mut` modifier, defines a mutable variable, and can be modified in the function. A parameter that is defined with the `ref` modifier, simulates a reference to the value passed to the function. It behaves similarly to a mutable variable, but mutating it also affects its value in the caller function. For example: `fn foo(mut x: u32, ref y: u32) { x *= 3; y = x + 1; }` `x` is a mutable parameter and `y` is a reference parameter. See below an example of calling it.

Return type

The return type defines the type of the value that is returned by the function when it is called. It is the type that appears in the signature after the `->`. Note that in Cairo, functions always return a value. When the function has no particular value it should return, it is common to return the [unit type](#) `()`. In this case, the return type can be omitted (including the `->`). In the example above, the return type is of type `T`.

Function body

The function body is the code that is executed when the function is called. It is enclosed by the curly braces `{...}` and consists of a list of 0 or more [statements](#), and an then an optional [expression](#) which is called the "tail expression". The statements are executed one after the other in the defined order. Then, if a tail expression exists, its evaluated value is returned by the function. If there is no tail expression, the function returns the [unit type](#) `()`. In the example above, the function body is `{ x + N }`, which consists of zero statements and a tail expression `x + N` whose value is the return value of the function.

Calling a function

See [Function calls](#).

[4.2 Use declarations](#) [4.4 Type aliases](#) [ð§](#)