A

Keywords

The following lists contain keywords that are reserved for current or future use by the Rust language. As such, they cannot be used as identifiers (except as raw identifiers, as we’ll discuss in “Raw Identifiers” on page XX). Identifiers are names of functions, variables, parameters, struct fields, modules, crates, constants, macros, static values, attributes, types, traits, or lifetimes.

Keywords Currently in Use

The following is a list of keywords currently in use, with their functionality described.

as

perform primitive casting, disambiguate the specific trait containing an item, or rename items in use statements

async

return a Future instead of blocking the current thread

await

suspend execution until the result of a Future is ready

break

exit a loop immediately

const

define constant items or constant raw pointers

continue

continue to the next loop iteration

crate

in a module path, refers to the crate root

dyn

dynamic dispatch to a trait object

else

fallback for if and if let control flow constructs

enum

define an enumeration

extern

link an external function or variable

false

Boolean false literal

fn

define a function or the function pointer type

for

loop over items from an iterator, implement a trait, or specify a higher-ranked lifetime

if

branch based on the result of a conditional expression

impl

implement inherent or trait functionality

in

part of for loop syntax

let

bind a variable

loop

loop unconditionally

match

match a value to patterns

mod

define a module

move

make a closure take ownership of all its captures

mut

denote mutability in references, raw pointers, or pattern bindings

pub

denote public visibility in struct fields, impl blocks, or modules

ref

bind by reference

return

return from function

Self

a type alias for the type we are defining or implementing

self

method subject or current module

static

global variable or lifetime lasting the entire program execution

struct

define a structure

super

parent module of the current module

trait

define a trait

true

Boolean true literal

type

define a type alias or associated type

union

define a union; is a keyword only when used in a union declaration

unsafe

denote unsafe code, functions, traits, or implementations

use

bring symbols into scope

where

denote clauses that constrain a type

while

loop conditionally based on the result of an expression

Keywords Reserved for Future Use

The following keywords do not yet have any functionality but are reserved by Rust for potential future use:

* abstract
* become
* box
* do
* final
* macro
* override
* priv
* try
* typeof
* unsized
* virtual
* yield

Raw Identifiers

Raw identifiers are the syntax that lets you use keywords where they wouldn’t normally be allowed. You use a raw identifier by prefixing a keyword with r#.

For example, match is a keyword. If you try to compile the following function that uses match as its name:

src/main.rs

fn match(needle: &str, haystack: &str) -> bool {

haystack.contains(needle)

}

you’ll get this error:

error: expected identifier, found keyword `match`

--> src/main.rs:4:4

|

4 | fn match(needle: &str, haystack: &str) -> bool {

| ^^^^^ expected identifier, found keyword

The error shows that you can’t use the keyword match as the function identifier. To use match as a function name, you need to use the raw identifier syntax, like this:

src/main.rs

fn r#match(needle: &str, haystack: &str) -> bool {

haystack.contains(needle)

}

fn main() {

assert!(r#match("foo", "foobar"));

}

This code will compile without any errors. Note the r# prefix on the function name in its definition as well as where the function is called in main.

Raw identifiers allow you to use any word you choose as an identifier, even if that word happens to be a reserved keyword. This gives us more freedom to choose identifier names, as well as lets us integrate with programs written in a language where these words aren’t keywords. In addition, raw identifiers allow you to use libraries written in a different Rust edition than your crate uses. For example, try isn’t a keyword in the 2015 edition but is in the 2018 edition. If you depend on a library that is written using the 2015 edition and has a try function, you’ll need to use the raw identifier syntax, r#try in this case, to call that function from your 2018 edition code. See Appendix E for more information on editions.