Panics the current thread.

This allows a program to terminate immediately and provide feedback to the caller of the program.

This macro is the perfect way to assert conditions in example code and in tests. `panic!` is closely tied with the `unwrap` method of both `Option` and `Result` enums. Both implementations call `panic!` when they are set to [`None`] or [`Err`] variants.

When using `panic!()` you can specify a string payload, that is built using the `format!` syntax. That payload is used when injecting the panic into the calling Rust thread, causing the thread to panic entirely.

The behavior of the default `std` hook, i.e. the code that runs directly after the panic is invoked, is to print the message payload to `stderr` along with the file/line/column information of the `panic!()` call. You can override the panic hook using `std::panic::set_hook()`. Inside the hook a panic can be accessed as a `&dyn Any + Send`, which contains either a `&str` or `String` for regular `panic!()` invocations. To panic with a value of another other type, `panic_any` can be used.

See also the macro [`compile_error!`], for raising errors during compilation.

When to use `panic!` vs `Result`

The Rust model of error handling groups errors into two major categories: recoverable and unrecoverable errors. For a recoverable error, such as a file not found error, it's reasonable to report the problem to the user and retry the operation. Unrecoverable errors are always symptoms of bugs, like trying to access a location beyond the end of an array.

The Rust language and standard library provides `Result` and `panic!` as parts of two complementary systems for representing, reporting, propagating, reacting to, and discarding errors for in these two categories.

The `panic!` macro is provided to represent unrecoverable errors, whereas the `Result` enum is provided to represent recoverable errors. For more detailed information about error handling check out the book or the `std::result` module docs.

Current implementation

If the main thread panics it will terminate all your threads and end your program with code `101`.

Examples

```
# #![allow(unreachable_code)]
panic!();
panic!("this is a terrible mistake!");
panic!("this is a {} {message}", "fancy", message = "message");
std::panic::panic_any(4); // panic with the value of 4 to be collected elsewhere
```