> For now, this reference is a best-effort document. We strive for validity and completeness, but are not yet there. In the future, the docs and lang teams will work together to figure out how best to do this. Until then, this is a best-effort attempt. If you find something wrong or missing, file an issue or send in a pull request.

Behavior considered undefined

Rust code, including within `unsafe` blocks and `unsafe` functions is incorrect if it exhibits any of the behaviors in the following list. It is the programmer's responsibility when writing `unsafe` code that it is not possible to let `safe` code exhibit these behaviors.

- Data races.
- Dereferencing a null or dangling raw pointer.
- Unaligned pointer reading and writing outside of `read_unaligned` and `write_unaligned`.
- Reads of undef (uninitialized) memory.
- Breaking the pointer aliasing rules on accesses through raw pointers; a subset of the rules used by C.
- `&mut T` and `&T` follow LLVM's scoped noalias model, except if the `&T` contains an `UnsafeCell<U>`.
- Mutating non-mutable data — that is, data reached through a shared reference or data owned by a `let` binding), unless that data is contained within an `UnsafeCell<U>`.
- Invoking undefined behavior via compiler intrinsics:
  - Indexing outside of the bounds of an object with `offset` with the exception of one byte past the end of the object.
  - Using `std::ptr::copy_nonoverlapping_memory`, a.k.a. the `memcpy32` and `memcpy64` intrinsics, on overlapping buffers.
- Invalid values in primitive types, even in private fields and locals:
  - Dangling or null references and boxes.
  - A value other than `false` (`0`) or `true` (`1`) in a `bool`.
  - A discriminant in an `enum` not included in the type definition.
  - A value in a `char` which is a surrogate or above `char::MAX`.
  - Non-UTF-8 byte sequences in a `str`.