`await` has been used outside `async` function or `async` block.

Erroneous code example:

```
# use std::pin::Pin;
# use std::future::Future;
# use std::task::{Context, Poll};
#
# struct WakeOnceThenComplete(bool);
#
# fn wake_and_yield_once() -> WakeOnceThenComplete {
#     WakeOnceThenComplete(false)
# }
#
# impl Future for WakeOnceThenComplete {
#     type Output = ();
#     fn poll(mut self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<()> {
#         if self.0 {
#             Poll::Ready(())
#         } else {
#             cx.waker().wake_by_ref();
#             self.0 = true;
#             Poll::Pending
#         }
#     }
# }
#
fn foo() {
    wake_and_yield_once().await // `await` is used outside `async` context
}
```

`await` is used to suspend the current computation until the given future is ready to produce a value. So it is legal only within an `async` context, like an `async` function or an `async` block.

```
# use std::pin::Pin;
# use std::future::Future;
# use std::task::{Context, Poll};
#
# struct WakeOnceThenComplete(bool);
#
# fn wake_and_yield_once() -> WakeOnceThenComplete {
#     WakeOnceThenComplete(false)
# }
#
# impl Future for WakeOnceThenComplete {
#     type Output = ();
#     fn poll(mut self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<()> {
#         if self.0 {
#             Poll::Ready(())
#         } else {
#             cx.waker().wake_by_ref();
#             self.0 = true;
#             Poll::Pending
#         }
#     }
# }
#
async fn foo() {
    wake_and_yield_once().await // `await` is used within `async` function
}

fn bar(x: u8) -> impl Future<Output = u8> {
    async move {
        wake_and_yield_once().await; // `await` is used within `async` block
        x
    }
}
```