This error occurs because a borrow in a generator persists across a yield point.

Erroneous code example:

```
# #![feature(generators, generator_trait, pin)]
# use std::ops::Generator;
# use std::pin::Pin;
let mut b = || {
    let a = &String::new(); // <-- This borrow...
    yield (); // ...is still in scope here, when the yield occurs.
    println!("{}", a);
};
Pin::new(&mut b).resume(());
```

At present, it is not permitted to have a yield that occurs while a borrow is still in scope. To resolve this error, the borrow must either be "contained" to a smaller scope that does not overlap the yield or else eliminated in another way. So, for example, we might resolve the previous example by removing the borrow and just storing the integer by value:

```
# #![feature(generators, generator_trait, pin)]
# use std::ops::Generator;
# use std::pin::Pin;
let mut b = || {
    let a = 3;
    yield ();
    println!("{}", a);
};
Pin::new(&mut b).resume(());
```

This is a very simple case, of course. In more complex cases, we may wish to have more than one reference to the value that was borrowed -- in those cases, something like the `Rc` or `Arc` types may be useful.

This error also frequently arises with iteration:

```
# #![feature(generators, generator_trait, pin)]
# use std::ops::Generator;
# use std::pin::Pin;
let mut b = || {
  let v = vec![1,2,3];
  for &x in &v { // <-- borrow of `v` is still in scope...
    yield x; // ...when this yield occurs.
  }
};
Pin::new(&mut b).resume(());
```

Such cases can sometimes be resolved by iterating "by value" (or using `into_iter()`) to avoid borrowing:

```
# #![feature(generators, generator_trait, pin)]
# use std::ops::Generator;
# use std::pin::Pin;
let mut b = || {
```

```
  let v = vec![1,2,3];
  for x in v { // <-- Take ownership of the values instead!
    yield x; // <-- Now yield is OK.
  }
};
Pin::new(&mut b).resume(());
```

If taking ownership is not an option, using indices can work too:

```
# #![feature(generators, generator_trait, pin)]
# use std::ops::Generator;
# use std::pin::Pin;
let mut b = || {
  let v = vec![1,2,3];
  let len = v.len(); // (*)
  for i in 0..len {
    let x = v[i]; // (*)
    yield x; // <-- Now yield is OK.
  }
};
Pin::new(&mut b).resume(());

// (*) -- Unfortunately, these temporaries are currently required.
// See <https://github.com/rust-lang/rust/issues/43122>.
```