Note: this error code is no longer emitted by the compiler.

This error occurs when an attempt is made to mutate or mutably reference data that a closure has captured immutably.

Erroneous code example:

```
// Accepts a function or a closure that captures its environment immutably.
// Closures passed to foo will not be able to mutate their closed-over state.
fn foo<F: Fn()>(f: F) { }

// Attempts to mutate closed-over data. Error message reads:
// `cannot assign to data in a captured outer variable...`
fn mutable() {
    let mut x = 0u32;
    foo(|| x = 2);
}

// Attempts to take a mutable reference to closed-over data.  Error message
// reads: `cannot borrow data mutably in a captured outer variable...`
fn mut_addr() {
    let mut x = 0u32;
    foo(|| { let y = &mut x; });
}
```

The problem here is that foo is defined as accepting a parameter of type `Fn`. Closures passed into foo will thus be inferred to be of type `Fn`, meaning that they capture their context immutably.

If the definition of `foo` is under your control, the simplest solution is to capture the data mutably. This can be done by defining `foo` to take FnMut rather than Fn:

```
fn foo<F: FnMut()>(f: F) { }
```

Alternatively, we can consider using the `Cell` and `RefCell` types to achieve interior mutability through a shared reference. Our example's `mutable` function could be redefined as below:

```
use std::cell::Cell;

fn foo<F: Fn()>(f: F) { }

fn mutable() {
    let x = Cell::new(0u32);
    foo(|| x.set(2));
}
```

You can read more in the API documentation for Cell.