This error indicates that a lifetime is missing from a type. If it is an error inside a function signature, the problem may be with failing to adhere to the lifetime elision rules (see below).

Erroneous code examples:

```
struct Foo1 { x: &bool }
              // ^ expected lifetime parameter
struct Foo2<'a> { x: &'a bool } // correct

struct Bar1 { x: Foo2 }
              // ^^^^ expected lifetime parameter
struct Bar2<'a> { x: Foo2<'a> } // correct

enum Baz1 { A(u8), B(&bool), }
                  // ^ expected lifetime parameter
enum Baz2<'a> { A(u8), B(&'a bool), } // correct

type MyStr1 = &str;
           // ^ expected lifetime parameter
type MyStr2<'a> = &'a str; // correct
```

Lifetime elision is a special, limited kind of inference for lifetimes in function signatures which allows you to leave out lifetimes in certain cases. For more background on lifetime elision see the book.

The lifetime elision rules require that any function signature with an elided output lifetime must either have:

- exactly one input lifetime
- or, multiple input lifetimes, but the function must also be a method with a `&self` or `&mut self` receiver

In the first case, the output lifetime is inferred to be the same as the unique input lifetime. In the second case, the lifetime is instead inferred to be the same as the lifetime on `&self` or `&mut self`.

Here are some examples of elision errors:

```
// error, no input lifetimes
fn foo() -> &str { }

// error, `x` and `y` have distinct lifetimes inferred
fn bar(x: &str, y: &str) -> &str { }

// error, `y`'s lifetime is inferred to be distinct from `x`'s
fn baz<'a>(x: &'a str, y: &str) -> &str { }
```