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

A lifetime of a returned value does not outlive the function call.

Erroneous code example:

```
fn prefix<'a>(
    words: impl Iterator<Item = &'a str>
) -> impl Iterator<Item = String> { // error!
    words.map(|v| format!("foo-{}", v))
}
```

To fix this error, make the lifetime of the returned value explicit:

```
fn prefix<'a>(
    words: impl Iterator<Item = &'a str> + 'a
) -> impl Iterator<Item = String> + 'a { // ok!
    words.map(|v| format!("foo-{}", v))
}
```

The `impl Trait` feature in this example uses an implicit `'static` lifetime restriction in the returned type. However the type implementing the `Iterator` passed to the function lives just as long as `'a`, which is not long enough.

The solution involves adding lifetime bound to both function argument and the return value to make sure that the values inside the iterator are not dropped when the function goes out of the scope.

An alternative solution would be to guarantee that the `Item` references in the iterator are alive for the whole lifetime of the program.

```
fn prefix(
    words: impl Iterator<Item = &'static str>
) -> impl Iterator<Item = String> {  // ok!
    words.map(|v| format!("foo-{}", v))
}
```

A similar lifetime problem might arise when returning closures:

```
fn foo(
    x: &mut Vec<i32>
) -> impl FnMut(&mut Vec<i32>) -> &[i32] { // error!
    |y| {
        y.append(x);
        y
    }
}
```

Analogically, a solution here is to use explicit return lifetime and move the ownership of the variable to the closure.

```
fn foo<'a>(
    x: &'a mut Vec<i32>
) -> impl FnMut(&mut Vec<i32>) -> &[i32] + 'a { // ok!
```

```
    move |y| {
        y.append(x);
        y
    }
}
```

To better understand the lifetime treatment in the `impl Trait`, please see the RFC 1951.