Something other than a type has been used when one was expected.

Erroneous code examples:

```
enum Dragon {
    Born,
}

fn oblivion() -> Dragon::Born { // error!
    Dragon::Born
}

const HOBBIT: u32 = 2;
impl HOBBIT {} // error!

enum Wizard {
    Gandalf,
    Saruman,
}

trait Isengard {
    fn wizard(_: Wizard::Saruman); // error!
}
```

In all these errors, a type was expected. For example, in the first error, if we want to return the `Born` variant from the `Dragon` enum, we must set the function to return the enum and not its variant:

```
enum Dragon {
    Born,
}

fn oblivion() -> Dragon { // ok!
    Dragon::Born
}
```

In the second error, you can't implement something on an item, only on types. We would need to create a new type if we wanted to do something similar:

```
struct Hobbit(u32); // we create a new type

const HOBBIT: Hobbit = Hobbit(2);
impl Hobbit {} // ok!
```

In the third case, we tried to only expect one variant of the `Wizard` enum, which is not possible. To make this work, we need to using pattern matching over the `Wizard` enum:

```
enum Wizard {
    Gandalf,
    Saruman,
}
```

```
trait Isengard {
    fn wizard(w: Wizard) { // ok!
        match w {
            Wizard::Saruman => {
                // do something
            }
            _ => {} // ignore everything else
        }
    }
}
```