An assignment operator was used on a non-place expression.

Erroneous code examples:

```
struct SomeStruct {
    x: i32,
    y: i32,
}

const SOME_CONST: i32 = 12;

fn some_other_func() {}

fn some_function() {
    SOME_CONST = 14; // error: a constant value cannot be changed!
    1 = 3; // error: 1 isn't a valid place!
    some_other_func() = 4; // error: we cannot assign value to a function!
    SomeStruct::x = 12; // error: SomeStruct a structure name but it is used
                        //        like a variable!
}
```

The left-hand side of an assignment operator must be a place expression. A place expression represents a memory location and can be a variable (with optional namespacing), a dereference, an indexing expression or a field reference.

More details can be found in the Expressions section of the Reference.

And now let's give working examples:

```
struct SomeStruct {
    x: i32,
    y: i32,
}
let mut s = SomeStruct { x: 0, y: 0 };

s.x = 3; // that's good !

// ...

fn some_func(x: &mut i32) {
    *x = 12; // that's good !
}
```