Physical Equality

OCaml has two equality operators, physical equality and structural equality. The documentation of `Pervasives.(==)` explains physical equality:

> `e1 == e2` tests for physical equality of `e1` and `e2`. On mutable types such as references, arrays, byte sequences, records with mutable fields and objects with mutable instance variables, `e1 == e2` is `true` if and only if physical modification of `e1` also affects `e2`. On non-mutable types, the behavior of `( == )` is implementation-dependent; however, it is guaranteed that `e1 == e2` implies `compare e1 e2 = 0`.

One interpretation could be that `==` should be used only when comparing refs (and other mutable data types) to see whether they point to the same location in memory. Otherwise, don't use `==`.

Structural equality is also explained in the documentation of `Pervasives.(=)`:

> `e1 = e2` tests for structural equality of `e1` and `e2`. Mutable structures (e.g. references and arrays) are equal if and only if their current contents are structurally equal, even if the two mutable objects are not the same physical object. Equality between functional values raises `Invalid_argument`. Equality between cyclic data structures may not terminate.

Structural equality is usually what you want to test. For refs, it checks whether the contents of the memory location are equal, regardless of whether they are the same location.

The negation of physical equality is `!=`, and the negation of structural equality is `<>`. This can be hard to remember.

Here are some examples involving equality and refs to illustrate the difference between structural equality (`=`) and physical equality (`==`):

```
# let r1 = ref 3110;;
val r1 : int ref = {contents = 3110}

# let r2 = ref 3110;;
val r2 : int ref = {contents = 3110}

# r1 == r1;;
- : bool = true

# r1 == r2;;
- : bool = false

# r1 != r2;;
- : bool = true

# r1 = r1;;
- : bool = true

# r1 = r2;;
- : bool = true

# r1 <> r2;;
- : bool = false

# ref 3110 <> ref 2110;;
- : bool = true
```