Next: Mutability, Previous: Type Predicates, Up: Lisp Data Types [Contents][Index]

2.8 Equality Predicates

Here we describe functions that test for equality between two objects. Other functions test equality of contents between objects of specific types, e.g., strings. For these predicates, see the appropriate chapter describing the data type.

Function: eq *object1 object2*

This function returns `t` if *object1* and *object2* are the same object, and `nil` otherwise.

If *object1* and *object2* are symbols with the same name, they are normally the same object—but see Creating Symbols for exceptions. For other non-numeric types (e.g., lists, vectors, strings), two arguments with the same contents or elements are not necessarily `eq` to each other: they are `eq` only if they are the same object, meaning that a change in the contents of one will be reflected by the same change in the contents of the other.

If *object1* and *object2* are numbers with differing types or values, then they cannot be the same object and `eq` returns `nil`. If they are fixnums with the same value, then they are the same object and `eq` returns `t`. If they were computed separately but happen to have the same value and the same non-fixnum numeric type, then they might or might not be the same object, and `eq` returns `t` or `nil` depending on whether the Lisp interpreter created one object or two.

```
(eq 'foo 'foo)
     ⇒ t


(eq ?A ?A)
     ⇒ t


(eq 3.0 3.0)
     ⇒ t or nil
;; Equal floats may or may not be the same object.


(eq (make-string 3 ?A) (make-string 3 ?A))
     ⇒ nil


(eq "asdf" "asdf")
     ⇒ t or nil
;; Equal string constants or may not be the same object.


(eq '(1 (2 (3))) '(1 (2 (3))))
     ⇒ nil


(setq foo '(1 (2 (3))))
     ⇒ (1 (2 (3)))
(eq foo foo)
     ⇒ t
(eq foo '(1 (2 (3))))
     ⇒ nil
```

```
(eq [(1 2) 3] [(1 2) 3])
     ⇒ nil


(eq (point-marker) (point-marker))
     ⇒ nil
```

The `make-symbol` function returns an uninterned symbol, distinct from the symbol that is used if you write the name in a Lisp expression. Distinct symbols with the same name are not `eq`. See Creating Symbols.

```
(eq (make-symbol "foo") 'foo)
     ⇒ nil
```

The Emacs Lisp byte compiler may collapse identical literal objects, such as literal strings, into references to the same object, with the effect that the byte-compiled code will compare such objects as `eq`, while the interpreted version of the same code will not. Therefore, your code should never rely on objects with the same literal contents being either `eq` or not `eq`, it should instead use functions that compare object contents such as `equal`, described below. Similarly, your code should not modify literal objects (e.g., put text properties on literal strings), since doing that might affect other literal objects of the same contents, if the byte compiler collapses them.

Function: equal *object1 object2*

This function returns `t` if *object1* and *object2* have equal components, and `nil` otherwise. Whereas `eq` tests if its arguments are the same object, `equal` looks inside nonidentical arguments to see if their elements or contents are the same. So, if two objects are `eq`, they are `equal`, but the converse is not always true.

```
(equal 'foo 'foo)
     ⇒ t


(equal 456 456)
     ⇒ t


(equal "asdf" "asdf")
     ⇒ t
(eq "asdf" "asdf")
     ⇒ nil


(equal '(1 (2 (3))) '(1 (2 (3))))
     ⇒ t
(eq '(1 (2 (3))) '(1 (2 (3))))
     ⇒ nil


(equal [(1 2) 3] [(1 2) 3])
     ⇒ t
(eq [(1 2) 3] [(1 2) 3])
     ⇒ nil


(equal (point-marker) (point-marker))
     ⇒ t
```

```
(eq (point-marker) (point-marker))
     ⇒ nil
```

Comparison of strings is case-sensitive, but does not take account of text properties—it compares only the characters in the strings. See Text Properties. Use `equal-including-properties` to also compare text properties. For technical reasons, a unibyte string and a multibyte string are `equal` if and only if they contain the same sequence of character codes and all these codes are in the range 0 through 127 (ASCII).

```
(equal "asdf" "ASDF")
     ⇒ nil
```

However, two distinct buffers are never considered `equal`, even if their textual contents are the same.

For `equal`, equality is defined recursively; for example, given two cons cells *x* and *y*, `(equal x y)` returns `t` if and only if both the expressions below return `t`:

```
(equal (car x) (car y))
(equal (cdr x) (cdr y))
```

Comparing circular lists may therefore cause deep recursion that leads to an error, and this may result in counterintuitive behavior such as `(equal a b)` returning `t` whereas `(equal b a)` signals an error.

Function: equal-including-properties *object1 object2*

This function behaves like `equal` in all cases but also requires that for two strings to be equal, they have the same text properties.

```
(equal "asdf" (propertize "asdf" 'asdf t))
     ⇒ t
(equal-including-properties "asdf"
                            (propertize "asdf" 'asdf t))
     ⇒ nil
```

Next: Mutability, Previous: Type Predicates, Up: Lisp Data Types [Contents][Index]