Special Operator **LET, LET***

Syntax:

let *({var | (var [init-form])}*) declaration* form* => result**

let* *({var | (var [init-form])}*) declaration* form* => result**

Arguments and Values:

var---a *symbol*.

init-form---a *form*.

declaration---a **declare** *expression*; not evaluated.

form---a *form*.

results---the *values* returned by the *forms*.

Description:

let and **let*** create new variable *bindings* and execute a series of *forms* that use these *bindings*. **let** performs the *bindings* in parallel and **let*** does them sequentially.

The form

```
 (let ((var1 init-form-1)
       (var2 init-form-2)
       ...
       (varm init-form-m))
   declaration1
   declaration2
   ...
   declarationp
   form1
   form2
   ...
   formn)
```

first evaluates the expressions *init-form-1*, *init-form-2*, and so on, in that order, saving the resulting values. Then all of the variables *varj* are bound to the corresponding values; each *binding* is lexical unless there is a **special** declaration to the contrary. The expressions *formk* are then evaluated in order; the values of all but the last are discarded (that is, the body of a **let** is an *implicit progn*).

let* is similar to **let**, but the *bindings* of variables are performed sequentially rather than in parallel. The expression for the *init-form* of a *var* can refer to *vars* previously bound in the **let***.

The form

```
(let* ((var1 init-form-1)
       (var2 init-form-2)
       ...
       (varm init-form-m))
  declaration1
  declaration2
  ...
  declarationp
  form1
  form2
  ...
  formn)
```

first evaluates the expression *init-form-1*, then binds the variable *var1* to that value; then it evaluates *init-form-2* and binds *var2*, and so on. The expressions *formj* are then evaluated in order; the values of all but the last are discarded (that is, the body of **let*** is an implicit **progn**).

For both **let** and **let***, if there is not an *init-form* associated with a *var*, *var* is initialized to **nil**.

The special form **let** has the property that the *scope* of the name binding does not include any initial value form. For **let***, a variable's *scope* also includes the remaining initial value forms for subsequent variable bindings.

Examples:

```
(setq a 'top) =>  TOP
(defun dummy-function () a) =>  DUMMY-FUNCTION
(let ((a 'inside) (b a))
   (format nil "~S ~S ~S" a b (dummy-function))) =>  "INSIDE TOP TOP"
(let* ((a 'inside) (b a))
   (format nil "~S ~S ~S" a b (dummy-function))) =>  "INSIDE INSIDE TOP"
(let ((a 'inside) (b a))
   (declare (special a))
   (format nil "~S ~S ~S" a b (dummy-function))) =>  "INSIDE TOP INSIDE"
```

The code

```
(let (x)
  (declare (integer x))
  (setq x (gcd y z))
  ...)
```

is incorrect; although `x` is indeed set before it is used, and is set to a value of the declared type *integer*, nevertheless `x` initially takes on the value **nil** in violation of the type declaration.

Affected By: None.

Exceptional Situations: None.

See Also:

progv

Notes: None.

The following X3J13 cleanup issues, *not part of the specification*, apply to this section:

- VARIABLE-LIST-ASYMMETRY:SYMMETRIZE

- DECLS-AND-DOC

Starting points Contents Chapter 1 Chapter 2 Master Index M N Symbol Index S T Glossary, *n.* Index of terms. x3j13 issues