ORACLE ACADEMY

Q. Print table of 17

Within a string a backslash (\) escapes the next character, causing it to be included in the string regardless of what it is.

op

"foo" → foo
"fo\o" → foo
"fo\\o" → fo\o
"fo\"o" → fo"o

To work b/w difference EQ & EQL

ORACLE ACADEMY

print "first: ~2d" a "second: ~2d" x y)

① x ← 25 → Print.

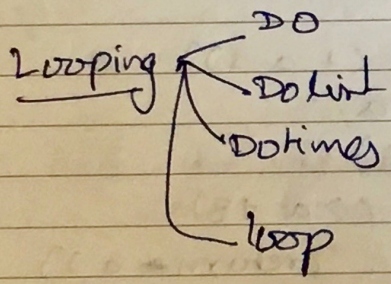
②

```
(setq a 10)
(loop
   (setq a (+ a 1))
   (write a)
   (terpri)
   (when (> a 18)
         (return a))
)
```

ORACLE ACADEMY

The concept of "progn" with multiple execution of statement of ~~yes~~ True and false.

(not nil) → T
(not (=1 1)) → NIL
(and (=1 2) (=3 3)) → NIL
(or (=1 2) (=3 3)) → T



ORACLE ACADEMY

"Dolist" loops across the ~~list~~ of items of a list, executing the loop body with a variable holding the successive items of the list.

(dolist (var list-form)
  (body-form*))

eg: (dolist (x '(1 2 3)) (print x))

O/P 1
2
3

If you want to break out of a Dolist loop before the end of a list, "RETURN" is used.

(dolist (x '(1 2 3)) (print x)
  (if (evenp x) (return)))

evenp - returns true if the integer is even.

oddp - returns true if the integer is odd.

ORACLE ACADEMY

```
(dotimes (x 20)
    (dotimes (y 20)
        (format t "~3d" (* (1+ x) (1+ y))))
    (format t "~%"))
```

```
(do (variable-definition*)
    (end-test form result-form*)
  statement*)
```

```
(do ((n 0 (1+ n))
     (cur 0 next)
     (next 1 (+ cur next)))
    ((= 10 n) cur))
```

```
(do ((i 0 (1+ i)))
    ((>= i 4))
  (print i))
```

ORACLE ACADEMY

variables

Common Lisp supports 2 kind of variables ← lexical (local)
dynamic (global)

"LET" — a special operator

```
(let (variable*)
  body-form)
```

```
(let ((x 10) (y 20) z) ...)
```

↳ binds 3 variable x, y, z with initial values 10, 20, NIL

```
(defun foo (x)
  (format t "parameter: ~a~%" x)
  (let ((x 2))
    (format t "outer let: ~a~%" x)
    (let ((x 3))
      (format t "inner LET: ~a~%" x))
    (format t "outer let: ~a~%" x))
  (format t "parameter: ~a~%" x))
```

ORACLE ACADEMY

LET*

in LET*, the initial value forms for each variable can refer to variables introduced earlier in the variable list.

```
(let* ((x 10)
       (y (+ x 10)))
  (list x y))
```

but not this

```
(let ((x 10)
      (y (+ x 10)))
  (list x y))
```

ORACLE ACADEMY

Common Lisp provides a number of type-specific equality predicates.

= compare numbers

CHAR= compare characters

EQ – Test for "object identity".
↳ should not be used to compare values that may be numbers of characters.

EQL – much better option

EQUAL similar to equal.
– It considers two strings equivalent if they contain the same character, ignoring differences in case. It also considers two character equivalent if they differ only in case.

Numbers are equivalent under EQUALP if they represent the same mathematical value. (equalp 1 1.0) - T

ORACLE ACADEMY

Common lisp provides two ways to create global variables < DEFVAR / DEFPARAMETER

Global variable are conventionally named with names that start and end with *.

eg. (defvar *count* 0
"count of widget made so far")

Binding examples

ORACLE ACADEMY

```
(defvar *x* 10)
(defun foo () (format t "X = %d~%" *x*))
```

if we run (foo)
o/p X:10

But when we use LET to create a new binding that shadows the global binding. like

```
(let ((*x* 20)) (foo))
```

o/p X:20

Now, call foo again without let

```
(foo)
```

o/p X:20

again define

```
(defun bar ()
  (foo)
  (let ((*x* 20)) (foo))
  (foo))
```

o/p → (bar)
X:10
X:20
X:10

Redefining foo to include an assignment to *x*.

ORACLE ACADEMY

```
(defun foo ()
  (format t "Before Assignment ~18tX: ~d~%" *x*)
  (setf *x* (+ 1 *x*))
  (format t "After assignment ~18tX: ~d~%" *x*))
```

Calling foo as (foo)

```
Before assignment X: 10
After assignment X: 11
```

Now run bar

```
(bar)
Before Assignment  X: 11
After Assignment   X: 12
Before Assignment  X: 20
Aft                X: 21
Bef                X: 12
Aft                X: 13
```

ORACLE ACADEMY

The name of every variable defined with DEFVAR and DEFPARAMETER is automatically declared globally special.

Constants:

All constants are global and are defined DEFCONSTANT.

```
(defconstant name initial-value-form)
```

LISP programmes follow a naming convention of using names starting and ending with + for constants.

① `(defun foo (x) (setf x 10))`

②
```
(let ((y 20))
  (foo y)
  (print y))
```

will print 20, not 10 as it's the value of y that is passed to foo and in it is briefly the value of the variable x before the setf gives x a new value.

ORACLE ACADEMY

I can write :

```
(setf x 1)
(setf y 2)
```

as

```
(setf x 1 y 2)
```

To increment a decrement

```
(incf x) ≡ (setf x (+ x 1))
(decf x) ≡ (setf x (- x 1))
(incf x 10) ≡ (setf x (+ x 10))
```

A macro ROTATEF rotates the values b/w places.

(rotatef a b)

swaps the values of two variables and return NIL.

ORACLE ACADEMY

```
(loop for i from 1 to 10 collecting i)
→ (1 2 3 4 5 6 7 8 9 10)
```

```
(loop for x from 1 to 10 summing (expt x 2))
→ 385
```

exponential.

```
(loop for x across "the quick brown fox
jumps over the lazy dog"
counting (find x "aeiou"))
```

O/P → 11

Common example of read

```
(setq x (read))
```

area of circle:

```
(defun areaofcircle()
  (terpri)
  (print "Enter radius: ")
  (setq radius (read))
  (setq area (* 3.144 radius radius))
  (princ "Area ")
  (write area))
(Area of circle)
```

List of abbreviations 24/07/17

all present Gr-[