ORACLE ACADEMY

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

$x \in A \cup (B \cap C)$

$x \in A$ or $x \in (B \cap C)$

$\Rightarrow x \in A$ or $[x \in B$ and $x \in C]$

$\Rightarrow (x \in A$ or $x \in B)$ and $(x \in A$ or $x \in C$

Cardinal no

$|A \cup B| = |A| + |B| - |A \cap B|$

$|A - B| = |A| - |A \cap B|$

$|B - A| = |B| - |A \cap B|$

$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B|$
$- |B \cap C| - |A \cap C| + |A \cap B \cap C|$

ORACLE ACADEMY

LISP :-

1. Various LISP environments:
   - Emacs,
   - lispworks,
   - Cygwin
   - Eclipse w/ plugin

~~Power of~~ LISP uses
1. Numeric literals
2. characters – strings
3. Functions
   - fns
   - macros
   - special operators

Direct numeric literals don't need parenthesis

```
> 1.1
> 2
> 3/2
> "abcdef"
> (print "abc")
```

Numeric literals

ORACLE ACADEMY

```
> ; reduced rational #
  20/2
  10

> ; hex                    for hexadecimal
  #x a
   10

> ; binary
  # b 1111
    15

> 123e-3
  .123

> (= 1 1)
  T

> (<= 2 3)
  T                        its NIL also

> PI
  3.14159 2653...
> (exp 1)   => 2.7182817
```

Functions:

ORACLE ACADEMY

1. Fundamental building block for LISP programs.

→ for format see 2 page afterwards

```
> (defun addtwo (x)
    (+ x 2))

> (addtwo 5)
=> 7
```

Write a function two multiply two numbers

```
<(defun mult (x y)
    (* x y))

< (mult 10 3)
  30
```

① sum of 2 no./ sum of 3 no.

② Square of a no.

HISTORY of LISP

ORACLE ACADEMY

Created by John McCarthy in late 1950's

- main idea was to study computability from a functional programming standpoint
→ CLISP now supports object oriented, imperative and functional programming

<S-expression> ::= <atom> | <list>
<atom> ::= <no.> | <identifier>
<list> ::= (<S-expression>)
<identifier> ::= string of printable characters not including parenthesis

Fundamental data types

ORACLE ACADEMY

1. Atoms → An atom is a alphanumeric string, used either as a variable name or a data item.
2. list → is a sequence of elements, where each element is either an atom or a list

A list is a left (or) parenthesis, followed by zero or more S-expressions, followed by a right parenthesis.
→ An S-expression is an atom or a list

In 1965, McCarthy developed a fn. called "eval" used to evaluate other fn.
- literals are evaluated to themselves.
eg. if you type in 5, the interpreter will return 5.

Expressions that are calls to primitive fn. are evaluated as follows:
1. Each parameter is evaluated first.
2. The primitive fn. is applied to parameter values.
3. The result is displayed.

> (eval 6/4)
3/2

> (eval (+ 2 3))
5

> (eval (+ (+ 1 1) (+ 1 1 1)))
5

Q.
> (+ < 3 2)) → error

> (if (< 2 3) (print "Yes") (print "No"))
Yes

> '(x 1 "foo")
(x 1 "foo")

> (+ a) → error

> (+ 1)
1

> (dotimes (x 2) (print x))
0
1

> (sort '(1 2 3) #'>) → for ascending/descending
1

Define or defun* is used to bind a name to a value
(defun (square num) (* num num))

format:

(defun (function_name parameters)
  (expression(s)))

sqrt is the inbuilt function.

Binding names to values

(define pi 3.14)
(define twopi (* 2 pi))

→ Once these two expressions are typed into Lisp interpreter, typing pi will return 3.14.

→ Names consists of letters, digits and special characters (except parenthesis) but can't begin with digits.

If statement :-

(if condition then-form [else-form])

- The then-form and optional else-form are restricted to a single lisp form.

"cond" is a macro to handle multiple if statements

```
(cond
    (test-1 form*)
    :
    (test-n form*))
```

(1)
```
(if (= 1 1)
    "a"
    "b")
```

(2)
```
(if (= 1 2)
    (+ 0.5 0.5)
    (- 0.5 0.5))
```
→ 0

(3)
```
(if (= 1 2)
    (if (= 1 2)
        (+ 0.5 0.5)
        (- 0.5 0.5)))
```
NIL

```
> cond (t (print "a"))
       (t (print "b"))
       (t (print "c")))
"a"

> cond (nil (print "a"))
       (nil (print "b"))
       (t (print "c"))
       (t (print "d")))

> (unless nil
      (print "a")
      (print "b"))
```

setf :- Binds a value to a location / variable.

(setf z '(1 2 3)) => (1 2 3)

function to reverse list :

(defun myreverse (lis)
  (let

Ph

```
(defun my-reverse (lis)
  (let ((new-list nil))
    (dolist (item lis)
      (setf new-list (cons item new-list)))
    new-list))
```

- NIL is the name of empty list
- As a test, NIL means false
- NIL is both an atom and a list

Basic fn.

(1) CAR returns the head of a list

(2) CDR returns the tail of a list

(3) CONS inserts a new head into a list

(4) EQ compares two atoms for equality

(5) ATOM tests if its argument is an atom.

(NULL S) tests if S is the empty list.

(LISTP S) tests if S is a list.

LIST makes a list of its (evaluated) arguments
- (LIST 'A '(B C) 'D) returns (A (B C) D)
- (LIST (CDR '(A B)) 'C) returns ((B) C)

APPEND concatenates two lists
(APPEND '(A B) '((X) Y))

=> (A B (X) Y)

(CAR '(a b c)) => a

note: Not using ' will result in an error.

What is the o/p in case of CDR
(A B C)
((X Y) Z)
(X)
((NIL) NIL)
NIL

ORACLE ACADEMY

CONS takes two arguments
- The first argument can be any S-exp
- The second argument should be a list

- The result is a new list whose CAR is the first argument and whose CDR is the second.

→ obtaining the second element of a list
(first (rest '(a b c d)))

'first' returns the first element of a list
(first '(a b c d))

'rest' returns a list with the first element removed.

ORACLE ACADEMY

Quotes stops expression evaluation.

```
> (+ 1 2 3)
6
> '(+ 1 2 3)
```

To assign a value to a symbol (setq, set, setf)

```
>(setq x 3.0)        >x
3.0                  3.0
```

- setq is a special form of function (with two arguments);
- The first argument is a symbol which will not be evaluated.
- The second argument

ORACLE ACADEMY

Area of a circle:

```
(defconstant PI 3.141592)
(defun area-circle (rad)
  (terpri)
  (format t "Radius: ~5f" rad)
  (format t "~%Area: ~10f" (* PI rad rad)))

(area-circle 10)
```

t → short hand for standard output

~t directive is for tabulating.
~10t tells FORMAT to emit enough spaces to move to the tenth column before processing the next ~a.

ORACLE ACADEMY

← What it do?

① `(butlast '(a b c d e f g) 3)`

② `(cons 'a '(a b))` → concatenating

`(cons '(a b) '(a b))`

③ Getting the length of a list
`(length '(a b c))`

④ Reversing a list
`(reverse '(a b c))`

⑤ Getting the last element of a list
`(last '(a b c d))`

⑥ creating a new list
`(list '(a b c) '(d e f))`

Expression like
'hello
(setf n 5)
are complete stand-alone programs, not simply statements.

> (Union L1 L2) : returns the union of two list.

) (Intersection L1 L2): returns the intersection of two list.

) (set-difference L1 L2): returns the difference of two list

Some Datastructures: — ATOMS / LISTS

Other data structure:
Association list, arrays, structures, dotten pairs, fn. streams.

1. Association lists are just list of sub-lists

(setf sarah '((height 0.54) (weight 4.4))
key ↑ value.

2. Retrieving elements from an association list

(assoc <key> <association-list>)
(assoc 'height sarah)
» ans. height .54

3 Creating a lisp array:

(setf a (make-array '(3 4)
:initial-contents
'((a b c d) 2 3 4)
("this" 5 6 7)
(#\c 8 9 10))))

4. Retrieving an array element.
(setf value (aref a 2 3))
ans 10

ORACLE ACADEMY

· Changing a value

```
(setf (aref a 2 3) 12)
```

- Creating & manipulating a lisp structure

```
(
```

Program of factorial.

ORACLE ACADEMY

```
(define (factorial (n)
    (if (eq n 0) 1
        (* n (factorial (- n 1))))))
```

What will be the O/P

```
(car '(this is a list of symbols))
```
=> this

```
(cdr '(singleton))
```
=> () the empty list.

```
(list 'a 'b 'c 'd 'e)
```
=> (a b c d e)

equivalent to

```
(cons 'a (cons 'b (cons 'c (cons 'd (
        cons 'e '()))))))
```

```
((cons 'a '())
```
=> (a)