UNIT- VI

① Explain the following equality predicates using suitable example:

(i) Equal:

It is true if its arguements are structurally similar (isomorphic) objects. Two objects are equal if they are equal strings of equal characters, bit or list of equal objects.

For example:

```
> (equal (list 1 2 3) (list 1 2 3))
=> T

> (equal "sid" "sid")
=> T

> (equal "SID" "sid")
=> NIL.
```

(ii) EQ:

The eq(x, y) is true if and only if x and y are the same identical object. That means x and y are usually eq if and only if they address the same identical memory location.

For example:

```
> eq('a 'b)
=> NIL

> eq('a 'a)
=> T
```

(iii) EQL :

It is true if its arguments are eq, or if they are numbers of same types with same value or if they are character objects that represent the same character (the comparison is case sensitive)

for eg

```
>eq (10, 10)
⇒ T

>eq (10, 10.0)
⇒ NIL
```

(iv) = :

compares only numbers, regardless of type

for eg-

```
> (= 10, 10.0)
⇒ T
```

② Explain the following Number predicates with using suitable example:

(i) NUMBERP :

It takes one argument and returns true if the argument is number, otherwise returns NIL.

eg:

```
> (numberp 10)
=> T

> (numberp 'a)
=> NIL
```

(ii) ZEROP :
This predicates return true if argument is zero.
for eg.

```
> (zerop 0)
=> T
```

(iii) PLUSP :
This predicate returns true if the number is strictly greater than zero and false otherwise
for eg.

```
>(plusp 10)
=> T

>(plusp -10)
=> NIL
```

(iv) EVENP :
It takes one numeric argument. If the number is even it returns T, otherwise returns NIL.
for eg.

```
>(evenp 10)
=> T

>(evenp 11)
=> NIL
```

(v) ODDP : It takes one argument. If the number is odd then it return T otherwise it returns NIL
eg.

```
> (oddp, 10)
=> NIL

> (oddp 11)
=> T
```

③ LISP program to find factorial of n numbers using recursion concept.

```
(defun fact (n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

④ Comparison bet^n functional and logic programming

Functional programming	Logical programming
(i) It uses functions	(i) It uses predicates
(ii) functions can return a value	(ii) Predicate cannot return a value. It can be true or false
(iii) the variable denotes the concret value	(iii) variable does not need to refer a concrete value
(iv) It is normally used for modelling reasoning	(iv) It is normally used for modelling knowledge
(v) Recursion is heavily used	(v) Recursion is frequently used.
(vi) High performance	(vi) slow performance
(vii) eg. ML, HASKELL, LISP	(vii) eg. prolog and Datalog.

(5) write sequence of CAR's and CDR's that will pick the atom pear out of the following s-expression?

(i) (apple orange pear grapes)

⇒

(car (cdr (cdr '(apple orange pear grapes))))

(ii) ((apple orange) (pear grapes))

⇒

(car (car (cdr '((apple orange) (pear grapes)))))

(iii) (((apple) (orange) (pear) (grapes)))

⇒

(car (car (car (cdr (cdr '(apple) (orange)
((pear)) (((grapes))))))))

(6) Evaluate the following forms of LISP.

(i) (car (cdr '(1 2 3 4 5)))

⇒ 2

(ii) (car (cdr '(a b c) d e)))

⇒ (b c)

(iii) (car (cdr (cdr '(1 2 3 4 5 6 7 8))))

⇒ 3

⑥ Explain the following function with suitable example.

(i) CAR() :

The function in lisp retrieves the first element of a list. If the list is empty it returns 'NIL'.

eg-

> (CAR '(1 2 34 5))

=> 1

(ii) CDR() :

The 'CDR()' function in Lisp retrieves all but the first element of a list. When means discarding first element rest list is printed. If the list is empty or contain only one element, it returns 'NIL'.

for eg-

> (CDR '(1 2 3))

=> 23

(iii) FIRST() :

The 'FIRST()' Function in Lisp is synonymous with 'car()'. It retrieves the first element of a list.

for eg.

> (FIRST '(1 23 45))

=> 1

⑦ Explain features of LISP programming

⇒

(i) Lisp Syntax : S-expression.
- It is based on S-expressions, which are nested lines of atoms and other s-expressions.
- It's simple syntax facilitates easy parsing and manipulation of code as data.

(ii) Interactive Development Environment :
- List environment typically provide an interactive development environment (REPL - Read-Eval-Print Loop) that allows developers to evaluate expressions, define functions and interactively experiment with code.

(iii) Dynamic Typing :
LISP is dynamically typed, meaning that variable types are determined at runtime rather than compile time. This flexibility simplifies coding and allows for rapid prototyping and reducing the risk of memory leaks.

(iv) Recursion :
It is a fundamental technique in LISP programming. LISP encourages recursive solutions to problems, and its simple syntax facilitates ~~for~~ the implementation of recursive algorithms.

(v) Portability :
LISP programs are often highly portable across different platforms and implementations. Standardization efforts, such as common LISP standard, help ensure compatibility and interoperability bet^n different LISP dialects.

(vi) Garbage collection :

LISP typically features automatic garbage collection which manages memory allocation and deallocation, freeing programmers from manual memory mangement tasks and reducing the risk of memory leaks.

(vii) Functional programming :

LISP supports functional programming paradigms including first-class functions, higher-order functions and lexical closures. Functions can be assigned to variables, passed as arguements and return as values.

(8) Explain basic list manipulation in prolog.

⇒

(i) cons

It take two arguements, an element & a list and returns a list with the element inserted at the first place.

eg.

```
> (write (cons 'a' (b c)))
=> (a b c)
```

(ii) list

It takes any no. of arguments and returns a list with the arguments as member elements of the list.

eg.

```
> (write (list 'a '(b c) '(e f)))
=> ( a (b c) (e f))
```

(iii) append

It merges two or more list into one

eg-

```
> (write (append '(b c) '(e f) '(p q) '(g)))
=> (b c e f p q g)
```

(iv) last

It takes a list and returns a list containing the last element

eg

```
> (write (last '(a b c d (e f))))
=> (e f)
```

Page No.
Date

* Describe logical programing, Enlist it features Also list commonly used logical programing languages ?

→

* logical programming language
- logical programming languge are type of programing language where you express the logic of computation without describing its control flow.
- these languges are based on formal logic
- It based on fact's and Rule's.
- It is declarative programing languge.
- focus is on logic.
- logical programming language is suitable for complex relationships and rule based logic.
- It is used in AI, natural language processing and Knowlege representation.

* features of logical programing language :-

1. Declarative Nature
- It state what the problem rather than how to solve it.

2. Based on formal logic
- It uses formal logic than Normal logic

3. Rule's Based syntax
- programs consist of series of rules and facts.

4. Automatic Backtracking :
- If one path fails, It automatically tries alternative path's.

5. Query - Based Execution :-
- programs are executed by passing queries to the system.

6. Use Most posibilities :-
- It tries multiple path until find the solution.

* Logical programing language :-

1. prolog
- Programing in logic
- Use in Artifical intelligence.

2. Datalog
- subset of prolog focus on database queries.

3. Nercury
- Use in real-word application.

4. Answer set programming (ASP)
- from diffecult combination find solution.

5. GHC
- Guarded Horn Clauses

* Describe fuctional programming languge? features and commonly used fuctional programming language?

* fuctional programming language :-
- fuctional programming languages are a category of programing languages designed to handle symbolic computation and list processing
- It focus on mathematical fuction's
- fuctional programming use fuction's to build the software.
- fuctions are the main building block. we can pass them any other data as a arguments.
- data of fuctional programing is imutabale or not-changable.

* features :-

1. First-class and Higher-order fuctions:
- fuction's are first-class citizen's meaning they can be assigned to variable or variable parssed as argument.
- fuction can take other fuction as argument.

2. Pure fuction :-
- pure fuction that alway's produce same output on same input

3. Imutability :-
- code once created cannot be changed.

4. fuction combining :
- Building complex fuction by combing one can be increare code reusability.

5. Recurison :-
- Recurison's are used insted of loop's.

6. Reusability :-
- one fuction can pass as a argument to other fuction.

* commonly used fuctional programing lang :-

1. Haskell :
- It is statically type.

2. Lisp :
- It suport both procedure and fuctional

3. Erlang :
- use in distrubuted system.

4. F# :
- It is .NE languge.

5. Scala :
- Integrated fuctional and object oriented.

27]

→ i] Term :
- In Prolog a term is basic building block of data It can be constant, a variable or a compound term.
- constant are atomic data structure like numbers or atoms
  (e.g → '42', 'hello';
- variable are placeholders for values that are not yet knows or are to be determined (e.g : 'x' , 'y');
- compound terms are structures formed by combining other terms using fuctions and parantheses.
  (e.g : 'parent(John, mary)' , 'add (3,4)

ii] Fact :
- Fact in Prolog are statements that tell ~~assert~~ relationships between entities.
- Fact's are like simple truth that we can assume to be correct.
- In facts we just simply describe relestionship between things.
- Example :
  - "RAM like pizza"
  - This pieces of information that we assume to be true in our program.

3. Rules :
- Rules in prolog are logical statements that defines relationships or conditions based on which queries can be resolved.
- They consist of a head and a body separated by the ':-' (if) symbol.
- head specifies what the rule is defining while the body specifies the condition that must be true for the rule to be satisfy.

4. Goals :
- Goals in prolog represents the queries or goals that the programmer want the prolog interpreter to solve
- Goals are predicates or queries that the prolog system attempts to satisfy by finding values for the variable that make the predicate true.
- The prolog interpreter searches for solutions by matching the goal with availiable facts and rules in knowledge base.

25. structure.

→ In prolog, structures are used to group related pieces of data together under a single name.
- They are similar to objects in other programming language.

* Example.
- suppose we want to represent information about person, including their name, age, and gender we can define a structure called 'person' to group this info

```
person (john, 25, male)
person (mike, 22, male)
```

- In this structure.
- Each person is represented by fact with the structure 'person (Name, Age, Gender)
- john, and mike are names of the people.
- 25, 22 are age of the people
- male, male are geneder of person.

- To find the age of a person named 'john'

→ person (john, Age, _);