UNIT-3

AI PROGRAMING LANGUAGES

LIPS का परिचय

LIPS का पूर्ण रूप है **Logic In Production System**। यह एक ऐसा **rule-based AI development system** है जिसका उपयोग **Expert System** बनाने के लिए किया जाता है। Expert System वे कंप्यूटर प्रोग्राम होते हैं जो किसी मानव विशेषज्ञ की तरह निर्णय लेने, समस्या हल करने या सलाह देने की क्षमता रखते हैं।

LIPS एक ऐसा वातावरण (environment) है जहाँ ज्ञान (**Knowledge**) को **IF–THEN** नियमों (production rules) के रूप में लिखा जाता है। इन नियमों को चलाने के लिए LIPS के पास एक **Inference Engine**, **Working Memory**, और **Control Strategy** होती है। इसी कारण LIPS का उपयोग विशेष रूप से **ट्यूटर सिस्टम, निर्णय-समर्थन सिस्टम, मेडिकल डायग्नोसिस और troubleshooting** जैसे क्षेत्रों में किया जाता है।

DEFINATION परिभाषा

LIPS एक उत्पादन-प्रणाली आधारित (production system based) आर्टिफिशियल इंटेलिजेंस भाषा और इंटरप्रेटर है, जिसमें ज्ञान को IF–THEN नियमों के रूप में संग्रहित किया जाता है। LIPS का inference engine नियमों का मिलान (matching), चयन (selection) और निष्पादन (execution) करके निष्कर्ष निकालता है। यह forward chaining तथा backward chaining दोनों का समर्थन करता है। LIPS का उपयोग उन समस्याओं को हल करने में किया जाता है जहाँ प्रतीकात्मक तर्क (symbolic reasoning), निर्णय-निर्माण, ज्ञान-आधारित निष्कर्ष और विशेषज्ञ जैसा व्यवहार आवश्यक होता है।

LIPS के मुख्य घटक (Main Components)

✓ 1. Production Rules (नियम)

LIPS में ज्ञान IF–THEN वक्तव्यों में लिखा जाता है।
उदाहरण:

IF बुखार AND खांसी

THEN रोग = फ्लू

ये नियम किसी विशेषज्ञ के ज्ञान का प्रतिनिधित्व करते हैं।

✓ 2. Working Memory (कार्य-स्मृति)

यह वह स्थान है जहाँ वर्तमान तथ्यों (facts) को रखा जाता है।
उदाहरण:

temperature = high

cough = yes

इन्हीं तथ्यों के आधार पर नियम सक्रिय होते हैं।

✓ 3. Inference Engine (तर्क इंजन)

यह सिस्टम का "दिमाग" होता है।
यह तय करता है:

- कौन-सा नियम चलेगा
- कब चलेगा
- किस क्रम में चलेगा

Inference Engine दो प्रकार से काम कर सकता है:

● Forward Chaining

तथ्यों से शुरू होकर निष्कर्ष की ओर जाता है।

● Backward Chaining

निष्कर्ष से शुरू होकर आवश्यक तथ्यों को खोजता है।

✓ 4. Conflict Resolution (संघर्ष समाधान)

अगर कई नियम लागू हो सकते हैं, तो LIPS तय करता है कि किस नियम को प्राथमिकता मिलेगी।
इसमें देखा जाता है:

- नियम की प्राथमिकता
- तथ्य कितने नए हैं
- नियम कितना विशिष्ट है

✓ 5. Explanation Facility (स्पष्टीकरण सुविधा)

LIPS बता सकता है:

- उसने कौन-सा निर्णय क्यों लिया
- कौन से नियम चले

- कैसे निष्कर्ष निकाला

यह Expert System की एक महत्वपूर्ण विशेषता है।

LIPS की विशेषताएँ (Features)

✓ सरल और स्पष्ट सिंटैक्स

नियम आसानी से लिखे और पढ़े जा सकते हैं।

✓ मॉड्यूलर ज्ञान

नियम जोड़ने/हटाने पर सिस्टम पर कम प्रभाव पड़ता है।

✓ प्रतीकात्मक तर्क (Symbolic Reasoning)

यह मनुष्य जैसे निर्णय ले सकता है।

✓ Forward और Backward Chaining का समर्थन

समस्या को दोनों प्रकार से हल कर सकता है।

✓ Explainable AI

सिस्टम निर्णयों का तर्क दे सकता है।

LIPS के उपयोग (Applications)

✓ Expert Systems

- मेडिकल निदान
- मशीन की खराबी का पता लगाना
- कानूनी सलाह

✓ शिक्षण (Tutoring Systems)

AI आधारित शिक्षण और व्याख्या प्रणाली।

✓ निर्णय-समर्थन सिस्टम (Decision Support)

बैंकिंग, बीमा, उद्योग आदि में उपयोग।

✓ तर्क और योजना (Reasoning & Planning)

जैसे नीति-निर्माण, troubleshooting आदि।

LIPS के लाभ (Advantages)

- सीखना और उपयोग करना आसान
- ज्ञान को नियमों में संगठित करना सरल
- तेज़ rule processing
- symbolic reasoning के लिए उपयुक्त
- निर्णयों का स्पष्टीकरण (why/how) दे सकता है

LIPS के नुकसान (Disadvantages)

- आधुनिक बड़े AI सिस्टम के लिए उपयोगी नहीं
- Machine Learning या numerical AI में उपयोग नहीं होता
- लगभग केवल शिक्षा और शोध में प्रयुक्त

Basic List Manipulation Functions

AI, LISP, Scheme जैसी भाषाओं में **List (सूची)** सबसे महत्वपूर्ण डेटा स्ट्रक्चर है। सूची में संख्याएँ, प्रतीक (symbols), या दूसरी सूचियाँ रखी जा सकती हैं।

इन सूचियों को उपयोग में लेने के लिए कई List Manipulation Functions उपयोग किए जाते हैं। ये फ़ंक्शन सूची को पढ़ने, बदलने, जोड़ने, हटाने, या बनाने में मदद करते हैं।

1. CAR

CAR वह फ़ंक्शन है जो किसी सूची (list) के **पहले तत्व (first element)** को लौटाता है। यह सूची के "head" या "front" को प्राप्त करने के लिए उपयोग होता है। यदि सूची खाली हो, तो CAR का परिणाम अनिश्चित होता है।

AI और LISP में यह फ़ंक्शन सूची को छोटे भागों में तोड़कर तर्क या recursion लागू करने में बहुत उपयोगी होता है।

उदाहरण:

(car '(10 20 30 40))

Output: 10

2. CDR

लंबी परिभाषा:

CDR किसी सूची से **पहला तत्व हटाकर बाकी की पूरी सूची** लौटाता है। यह सूची की "tail" या "remaining part" को देता है। CDR का उपयोग सूची को step-by-step प्रोसेस करने में बहुत महत्वपूर्ण है।

उदाहरण:

(cdr '(10 20 30 40))

Output: (20 30 40)

3. CONS

CONS वह फ़ंक्शन है जो **किसी सूची की शुरुआत में एक नया तत्व जोड़कर नई सूची बनाता है**। CONS लिस्ट निर्माण (construction) का आधार है। recursive एल्गोरिद्म में नई सूचियाँ बनाने के लिए यही सबसे ज़्यादा उपयोग होता है।

उदाहरण:

(cons 5 '(10 20 30))

Output: (5 10 20 30)

4. LIST

LIST एक ऐसा फ़ंक्शन है जो दिए गए तत्वों की मदद से **नई पूरी सूची बनाता है**। CONS एक-एक तत्व जोड़ता है, जबकि LIST एक ही बार में पूरी सूची बना देता है।

उदाहरण:

(list 1 2 3 4)

Output: (1 2 3 4)

5. APPEND

APPEND दो या अधिक सूचियों को **जोड़कर एक एकल (single) सूची** बनाता है। यह मूल सूचियों को बदले बिना एक नई संयुक्त सूची लौटाता है। AI एल्गोरिद्म में यह path building, merging, और combining operations में उपयोगी है।

उदाहरण:

(append '(1 2) '(3 4 5))

Output: (1 2 3 4 5)

6. LENGTH

LENGTH सूची में मौजूद **कुल तत्वों की संख्या** लौटाता है। recursive प्रोग्राम में, list traversal में, और stopping condition तय करने में इसका उपयोग होता है।

उदाहरण:

(length '(5 10 15))

Output: 3

7. REVERSE

REVERSE एक नई सूची बनाता है जिसमें **सभी तत्व उल्टे क्रम (reverse order)** में होते हैं। यह backtracking, stack operations और normalization के लिए उपयोगी है।

उदाहरण:

(reverse '(1 2 3))

Output: (3 2 1)

8. NTH (या ELEMENT-AT)

NTH किसी सूची में दिए गए **इंडेक्स (position)** पर मौजूद तत्व को लौटाता है। अधिकतर स्थानों पर index 0 से शुरू होता है। यह बिना पूरी सूची traverse किए किसी तत्व को सीधे प्राप्त करने के काम आता है।

उदाहरण:

(nth 2 '(10 20 30 40))

Output: 30

9. MEMBER

MEMBER यह जांचता है कि **किसी सूची में कोई निश्चित तत्व मौजूद है या नहीं**। यदि मौजूद हो, तो उस तत्व से शुरू होने वाली उप-सूची (sublist) लौटाता है; नहीं होने पर NIL लौटाता है। इसे pattern matching और searching में उपयोग किया जाता है।

उदाहरण:

(member 30 '(10 20 30 40))

Output: (30 40)

10. MAPCAR या MAP

MAPCAR एक फ़ंक्शन को सूची के **हर तत्व पर लागू करता है** और परिणामों को एक नई सूची के रूप में लौटाता है। यह data transformation, processing और AI में uniform logic apply करने के लिए बहुत उपयोगी है।

उदाहरण:

(mapcar 'square '(1 2 3 4))

Output: (1 4 9 16)

इन फ़ंक्शनों का महत्व (Importance)

- सूची को तोड़ने, जोड़ने, या बदलने में उपयोग
- Recursive algorithms का आधार
- Symbolic AI और Expert Systems में ज़रूरी
- Knowledge representation और reasoning में उपयोग
- Tree traversal और pattern matching के लिए आवश्यक
- Functional programming का मूल

Short Notes (2–Mark Points)

- **CAR** – सूची का पहला तत्व देता है
- **CDR** – सूची का अवशिष्ट (बाकी हिस्सा) देता है
- **CONS** – शुरुआत में तत्व जोड़कर नई सूची बनाता है
- **LIST** – नई सूची तैयार करता है
- **APPEND** – अलग-अलग सूचियों को जोड़ता है
- **LENGTH** – कुल तत्वों की संख्या बताता है
- **REVERSE** – सूची को उल्टे क्रम में करता है
- **NTH** – खास स्थान का तत्व देता है
- **MEMBER** – तत्व की उपस्थिति जांचता है
- **MAPCAR** – फ़ंक्शन को हर तत्व पर लागू करता है

AI में Input, Output और Local Variables

Artificial Intelligence (कृत्रिम बुद्धिमत्ता) किसी भी समस्या को हल करने के लिए सबसे पहले **input** लेती है, उस पर नियम और logic लागू करके **processing** करती है, और अंत में **output** पैदा करती है। इस प्रक्रिया में AI प्रोग्राम अस्थायी रूप से कुछ डेटा को **local variables** में भी संग्रहीत करता है।

आइए तीनों को विस्तार से समझते हैं।

1. AI में Input (इनपुट)

परिभाषा (Definition)

AI में input वह डेटा, तथ्य, निर्देश या जानकारी है जिसे सिस्टम उपयोगकर्ता, सेंसर, फाइल या किसी अन्य स्रोत से प्राप्त करता है ताकि समस्या का समाधान शुरू किया जा सके।

- इनपुट **AI प्रोग्राम का प्रारंभिक बिंदु** होता है।
- बिना इनपुट के कोई AI सिस्टम निर्णय नहीं ले सकता।
- Input किसी भी प्रकार का हो सकता है:
    - उपयोगकर्ता का प्रश्न
      (जैसे: "सबसे छोटा रास्ता बताओ।")
    - नियम या तथ्य
      ("राम एक व्यक्ति है")
    - सेंसर डेटा
      (तापमान, इमेज, वीडियो)
    - डेटाबेस से लिया गया डेटा
    - लिस्ट या स्ट्रक्चर (LISP/Prolog में)
    - मशीन लर्निंग के लिए संख्या, इमेज, टेक्स्ट

Input का उद्देश्य

- समस्या का **विवरण** देना
- AI को **शुरुआती जानकारी** देना
- सिस्टम में **reasoning/learning** शुरू करना

2. AI में Output (आउटपुट)

परिभाषा

Output वह अंतिम परिणाम, निष्कर्ष, उत्तर, निर्णय या क्रिया है जो AI सिस्टम इनपुट को संसाधित करने के बाद उत्पन्न करता है।

- Output AI की **सोचने और समझने की क्षमता** का परिणाम है।
- यह कई प्रकार का हो सकता है:
    - कोई संख्यात्मक उत्तर
    - कोई सूची (List) या डेटा स्ट्रक्चर
    - कोई निर्णय
    - किसी बीमारी का निदान
    - किसी गेम में अगला कदम
    - स्पीच पहचान कर टेक्स्ट निकालना
    - मशीन लर्निंग मॉडल द्वारा की गई भविष्यवाणी
- Symbolic AI (LISP/Prolog) में आउटपुट अक्सर:
    - लॉजिकल परिणाम
    - नई लिस्ट
    - सत्य/असत्य
    - कोई निष्कर्ष

Output का उद्देश्य

- समस्या का **समाधान** प्रस्तुत करना
- नियमों और logic का **अंतिम प्रभाव** दिखाना
- उपयोगकर्ता या अन्य प्रोग्राम को **परिणाम** बताना

3. AI में Local Variables (लोकल वेरिएबल्स)

परिभाषा

Local variables वे अस्थायी (temporary) वेरिएबल्स होते हैं जिनका उपयोग किसी फ़ंक्शन, नियम, या एल्गोरिथ्म के अंदर किया जाता है और जो उसी हिस्से तक सीमित रहते हैं।

- लोकल वेरिएबल्स **अस्थायी डेटा** को स्टोर करने के लिए उपयोग किए जाते हैं।
- इनका अस्तित्व केवल उस समय तक रहता है जब तक फ़ंक्शन या नियम चल रहा होता है।
- इनका उपयोग AI में बहुत अधिक होता है जैसे:
    - recursion
    - pattern matching
    - list processing
    - search algorithms (DFS, BFS)
    - temporary calculations

मुख्य गुण

1. **सीमित दायरा (Scope)**
   जहाँ घोषित किए गए हैं, केवल वहीं तक उपयोग में आते हैं।
2. **अस्थायी जीवनकाल (Lifetime)**
   फ़ंक्शन पूरा होते ही समाप्त हो जाते हैं।
3. **AI अल्गोरिद्म में उपयोग**
   LISP, Prolog, Python, Search Algorithms, Parsing आदि में बहुत जरूरी हैं।

सरल उदाहरण (Conceptual Example)

```
path(start, end):
  temp_distance = ...   ← लोकल वेरिएबल
  best_route = ...      ← लोकल वेरिएबल
  return best_route
```

AI में List और Array

AI में डेटा को संगठित (organize), संग्रहित (store), और संसाधित (process) करने के लिए कई data structures उपयोग किए जाते हैं।
उनमें से **List** और **Array** सबसे महत्वपूर्ण हैं, खासकर **LISP**, **Prolog**, **Python**, और अन्य AI भाषाओं में।

AI में दोनों का उपयोग **knowledge representation**, **search algorithms**, **pattern matching**, **problem solving**, और **symbolic computation** में होता है।

1. LIST in AI (लिस्ट)

AI में **List** एक ऐसा ordered data structure है जिसमें कई elements को एक अनुक्रम (sequence) में संग्रहीत किया जाता है।
List बेहद लचीली (flexible) होती है–इसमें अलग-अलग प्रकार के डेटा (numbers, symbols, sub-lists, strings) एक साथ रखे जा सकते हैं।
AI में List का प्रयोग data को symbolic form में represent करने, reasoning करने, recursion लागू करने, और problem-solving में किया जाता है।

✓ 1. Ordered Collection

List में elements एक निश्चित क्रम (order) में रखे जाते हैं।
उदाहरण:
(A B C D) या (1 2 3 4)

✓ 2. Heterogeneous Data

List में अलग-अलग प्रकार का डेटा एक साथ हो सकता है:
(10 "Ram" TRUE (1 2 3))

✓ 3. Dynamic Size (आकार बदल सकता है)

List का size runtime पर बढ़ या घट सकता है।

✓ 4. Symbolic AI में मुख्य संरचना

LISP में लगभग पूरा प्रोग्राम **list** पर आधारित होता है।
AI में हर knowledge structure—tree, graph, frames—List से represent किया जा सकता है।

✓ 5. Sub-list (Nested Lists)

List के अंदर दूसरी List हो सकती है:
((A B) (C D) (E (F G)))

✓ 6. Recursion Friendly

अधिकांश AI समस्याएँ recursive techniques से हल होती हैं, जिसमें list perfect data structure है।

List Operations in AI

मुख्य operations हैं:

- **CAR** → first element निकालता है
- **CDR** → बाकी list निकालता है
- **CONS** → नया element जोड़ता है
- **APPEND** → दो lists को जोड़ता है
- **LENGTH** → size देता है

ये LISP में अत्यंत उपयोगी हैं।

Uses of List in AI

- Knowledge representation
- Pattern matching
- Natural language processing
- Search states representation
- Game tree, decision tree
- Grammar representation
- Symbolic computation

2. ARRAY in AI (ऐरे)

AI में **Array** एक ऐसा data structure है जिसमें समान प्रकार के elements को एक निश्चित आकार (fixed size) के continuous memory block में संग्रहित किया जाता है।
Array indexing के आधार पर सीधे (direct) data access प्रदान करता है, इसलिए गणनात्मक (computational) कार्यों और numerical AI tasks जैसे machine learning में इसका उपयोग अधिक होता है।

✓ 1. Fixed Size (स्थिर आकार)

Array का आकार तय होता है और runtime पर नहीं बदलता (कुछ भाषाओं में dynamic arrays होते हैं)।

✓ 2. Homogeneous Data

Array में सभी elements एक ही प्रकार के होते हैं:
जैसे integer array, float array आदि।

✓ 3. Fast Access (तेज़ पहुँच)

Indexing का उपयोग होने से किसी भी element को तुरंत access किया जा सकता है:

arr[0], arr[1], arr[2]

✓ 4. Continuous Memory Allocation

एक ही block में डेटा संग्रहीत होता है, इसलिए processing तेज होती है।

✓ 5. Numerical AI में महत्वपूर्ण

Machine learning algorithms में matrices, vectors, tensors — सभी arrays पर आधारित होते हैं।

Array का उपयोग AI में

- Matrix operations
- Neural networks (weights, bias arrays)
- Image processing (image = pixel array)
- Deep learning tensors
- Search grids
- Reinforcement learning environments

Difference Between List and Array (AI Perspective)

Feature	List	Array
Data Type	Heterogeneous (mix data)	Homogeneous (same type)
Size	Dynamic	Mostly fixed
Structure	Linked or nested	Continuous memory
Use in AI	Symbolic AI, LISP, Prolog	Numerical AI, ML, DL
Flexibility	Very flexible	Less flexible
Speed	Slower for random access	Faster for random access

1. “Hello World” प्रोग्राम

(print "Hello World")

व्याख्या:

print टेक्स्ट को स्क्रीन पर दिखाता है।

2. दो संख्याओं को जोड़ना

```
(+ 10 20)
```

व्याख्या:

LISP में ऑपरेटर सबसे पहले आता है → (operator operand1 operand2)

यह 10 + 20 का परिणाम देगा।

3. दो संख्याओं को घटाना

```
(- 50 25)
```

4. दो संख्याओं को गुणा करना

```
(* 5 6)
```

5. किसी संख्या का Square निकालना

```
(defun square (x)
 (* x x))
```

प्रयोग:

```
(square 5)
```

6. संख्या Positive है या नहीं

```
(defun positive? (x)
 (if (> x 0)
    "Positive"
    "Not Positive"))
```

7. Factorial प्रोग्राम (Recursion)

```
(defun fact (n)
 (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

प्रयोग:

(fact 5)

8. लिस्ट का पहला एलिमेंट निकालना

(car '(10 20 30 40))

आउटपुट:

10

9. लिस्ट के आगे एक नया एलिमेंट जोड़ना

(cons 5 '(10 20 30))

आउटपुट:

(5 10 20 30)

10. लिस्ट के सभी एलिमेंट्स का sum निकालना

(defun sum-list (lst)

(if (null lst)

0

(+ (car lst) (sum-list (cdr lst)))))

प्रयोग:

(sum-list '(1 2 3 4))

PROLOG का परिचय (Introduction to PROLOG)

PROLOG एक उच्च-स्तरीय (high-level) **logic programming language** है, जिसे विशेष रूप से **Artificial Intelligence (AI)** और **Computational Linguistics** में उपयोग किया जाता है। PROLOG का पूरा नाम **PROgramming in LOGic** है। यह भाषा परंपरागत programming languages (जैसे C, Java, Python) से बहुत अलग है, क्योंकि PROLOG में हम *कैसे करना है* (How to do) नहीं बताते, बल्कि *क्या करना है* (What to do) बताते हैं।

PROLOG एक ऐसी तर्क-आधारित (logic-based) प्रोग्रामिंग भाषा है जिसमें प्रोग्राम को facts (तथ्य), rules (नियम), और queries (प्रश्न) के रूप में लिखा जाता है। यह भाषा symbolic reasoning, pattern matching, knowledge representation और समस्या समाधान के लिए अत्यंत उपयुक्त है। PROLOG में प्रोग्रामिंग declarative शैली में की जाती है, जहाँ हम सिर्फ समस्या बताते हैं और

PROLOG अपने built-in inference engine और unification process का उपयोग करके स्वतः उत्तर निकालता है।

PROLOG की मूल अवधारणाएँ (Key Concepts)

Declarative Language (घोषणात्मक भाषा)

PROLOG में आप यह नहीं बताते कि समाधान कैसे निकलेगा, बल्कि आप यह बताते हैं कि:

- कौन-कौन से facts सही हैं
- कौन-कौन से rules लागू होते हैं
- और फिर system खुद reasoning करता है

इस कारण PROLOG AI में बहुत उपयोगी है।

Facts (तथ्य)

ये वे statements होते हैं जो हमेशा सत्य माने जाते हैं।

जैसे:

parent(ram, shyam).

male(ram).

female(gita).

Rules (नियम)

Rules किसी fact के आधार पर और facts बना सकते हैं।
इनमें condition होती है।

जैसे:

grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

मतलब -
यदि **X Y का parent है** और **Y Z का parent है**,
तो **X Z का grandparent है**।

Queries (प्रश्न)

User PROLOG से प्रश्न पूछता है, और PROLOG answer खोजता है।

जैसे:

?- parent(ram, shyam).

?- grandparent(ram, X).

Unification (एकीकरण)

PROLOG अपने reasoning system में *matching* तकनीक का उपयोग करता है जिसे **unification** कहा जाता है।

यह variables और values की तुलना करके सही उत्तर निकालता है।

Backtracking (उल्टा खोज तंत्र)

अगर PROLOG को किसी रास्ते से उत्तर नहीं मिलता तो वह वापस जाकर दूसरा रास्ता आज़माता है। इसे **backtracking** कहते हैं।

यह PROLOG की सबसे शक्तिशाली विशेषताओं में से एक है।

PROLOG का उपयोग कहाँ होता है?

PROLOG विशेष रूप से उपयोग होता है:

- Expert systems
- Natural Language Processing
- Knowledge-based systems
- Automated reasoning
- Robotics
- Intelligent tutoring systems
- Theorem proving
- Medical diagnosis systems

PROLOG की विशेषताएँ (Features)

- Logic-based programming
- Pattern matching
- Automatic backtracking
- Symbol processing
- Data representation using facts & rules

- Reasoning power (AI applications)
- Recursion का extensive उपयोग
- Extremely powerful for AI problems

PROLOG क्यों AI के लिए सबसे उपयुक्त है?

- इसमें मनुष्य की तरह तर्क (logical reasoning) होता है।
- Knowledge को facts और rules के रूप में आसानी से represent करता है।
- Automatic search और backtracking समस्याओं का समाधान आसान बनाते हैं।
- Natural language और symbolic data को बहुत अच्छे से संभालता है।