1. doua tipuri de recursivitate: directa sau indirecta

**Prolog**

1. Variabilele incep cu litera mare (simbolurile pt constante incep cu litere mici)
2. Variabila anonima in prolog = “\_”
3. Lista vida e singura lista considerata atom in prolog
4. Variabilele sunt folosite ca parte a unui proces de potrivire, nu ca un tip de stocare de informaţii
5. Structurile identice se potrivesc una cu alta (ex. p(a, b) cu p(a, b));
6. Doua variabile libere se potrivesc una cu alta (p(a, X) cu p(a, Y) daca X, Y - libere)
7. X=Y verifica daca X si Y pot fi unificate
8. ≤ se scrie =< ; ≥ se scrie >=
9. X==Y verifica daca sunt legate la aceeasi valoare
10. Utilizarea 2+4 nu e aceeași ca utilizarea lui 6
11. =:= testeaza egalitatea aritmetica; =\= - diferit
12. Predicat determininst - o singura solutie; nedeterminist - mai multe; acelasi predicat poate varia in functie de flux
13. findall(argumentul din predicat care se colecteaza, predicatul care se rezolva, lista colectoare)
    1. Ex. findall(Y, caut(X, Y), L) => L o sa contina toate solutiile pt Y dupa executia lui “caut”
14. Cap (head) lista = primul elem; coada (tail) = restul
15. Lista vida nu poate fi impartita in cap/coada
16. înaintea caracterului ‘|’ pot fi scrise mai multe elemente, nu doar primul
17. Nu se executa nici un predicat de dupa “fail”
18. Recomandat sa folosim cut dupa ce am verificat toate conditiile
19. “Is” merge cu numere, “=” cu liste
20. variabilele sunt locale (daca 2 functii contin un X, sunt diferiti si nu au legatura unul cu altul)
21. Cand avem o intrebare de genul: “ce face codul ?”, **depinde de flow model** (ex. [s3, p5](http://www.cs.ubbcluj.ro/~marianzsu/PLFEn/docs/PLF_Seminar3_en.pdf#page=5))
22. \+ neaga predicate (ex. \+ is\_list(L) )
23. number(X) verifica daca X e atom numeric

**Lisp**

1. in lisp nu exista nedeterminism
2. orice lista simplu inlantuita se poate reprezenta ca un arbore, dar invers nu
3. orice lista are o “pereche cu punct” echivalenta, dar invers nu
4. NIL e si lista (empty list) si atom (valoarea lui false)
5. If we want the first element of the list to not be considered a function (because for example is a simple list which is our parameter), we have to place an apostrophe in front of the list: ‘(1 2 3 4 5 6). If we write only the list, without the apostrophe, Lisp will give us an error: 1 is not a function name.
6. “and”, “or” nu sunt functii, sunt operatori speciali
7. defun poate redefini functiile standard (ex. car)
8. sunt situatii in care folosirea unei variabile colectoare creste complexitatea (daca trebuie sa adaugam la sf ei)
9. APPEND lucreaza de obicei cu copiile elementelor, mai putin ultimul element, pt ca nu legam nimic de el; daca vrem sa lucram cu elementele efective, folosim NCONC
10. daca o functie, pe langa calculul valorii sale, modifica si structurile de date din memorie => efect secundar (set, setq); efectul principal e ca returneaza ceva (se cheama functie tocmai daca returneaza ceva)
11. Setq nu evalueaza argumentele carora le dam valori, spre deosebire de set
12. (setq X '(a b)) => X = (a b)
    1. Apoi (setf (car x) 0) => X = (0 b)
13. setf este un macro, are efect distructiv
14. argumentele in LAMBDA sunt evaluate la apel; daca nu vrem asta, folosim QLAMBDA
15. nu putem folosi un lambda recursiv pt ca nu are nume
16. Lisp nu evaluează primul element dintr-o formă, ci numai îl aplică
17. O listă este totdeauna evaluată dacă acest lucru nu este oprit explicit (prin QUOTE), în schimb primul argument al oricărei liste nu este niciodată evaluat
18. mapcan / mapcon folosesc nconc, mapcar / maplist – list
19. (mapcan #'*f* L) <=> (apply #'append (mapcar #'f L))
20. &optional lucreaza cu un singur argument extra, &rest cu o lista