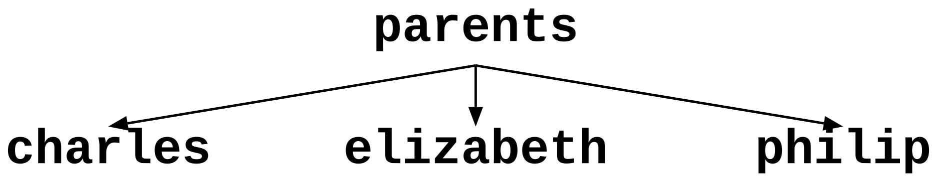
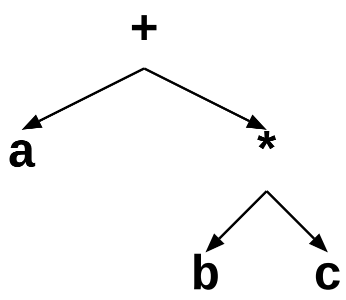
Tree Structure

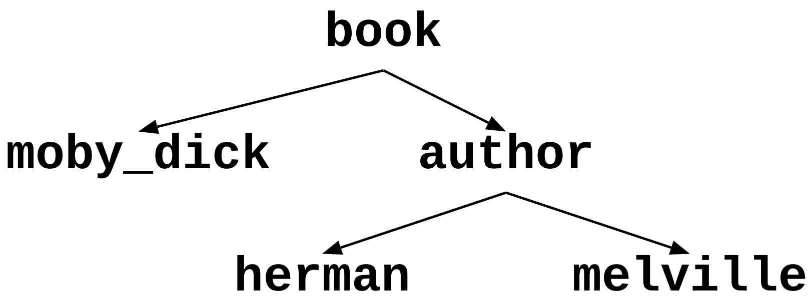
```
parents(charles,elizabeth,philip).
```



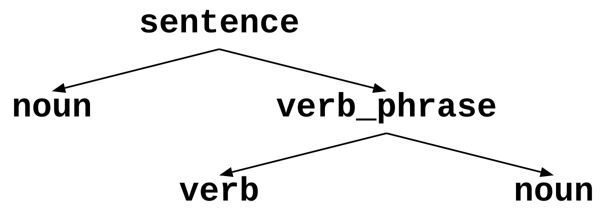

```
a+b*c
```




```
book(moby_dick,author(herman,melville)).
```

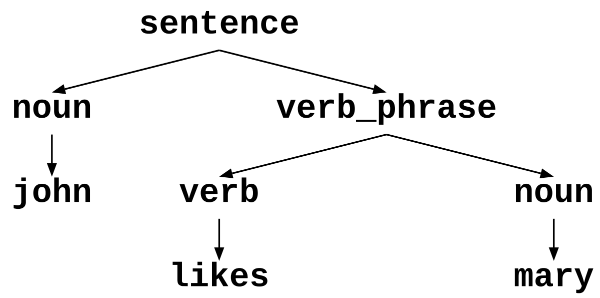


Parsing Sentences



john likes mary.

```
sentence(noun(john),verb_phrase(verb_phrase(likes),noun(mary)))).
```



Lists

Ordered Sequence of Elements
of any length

Can be thought of as a special type of tree
ala LISP (CONS, CDR)

Empty List

[]

One Element

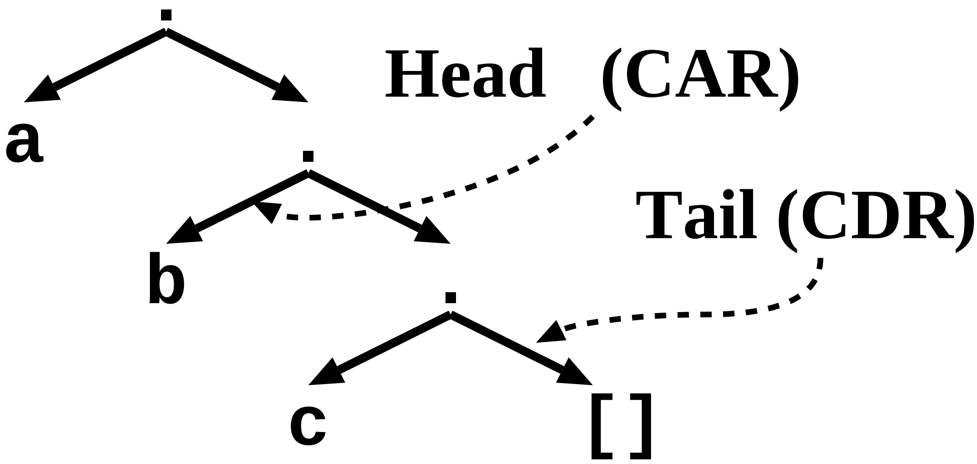
.(a,[])

Two Elements

.(a,.(b,[]))

List as Tree

`.(a,.(b,.(c,[])))`



PROLOG Lists

List Notation

$$[]$$
$$[a]$$
$$[a, b]$$
$$[a, b, c]$$

$$[a, [b, c, d], e, f, g]$$

Head and Tail

[a,b,c,d]	a	[b,c,d]
[a]	a	[]
[]	fails	fails
[[the,cat],sat]	[the,cat]	[sat]
[X+Y,x+y]	X+Y	[x+y]

Head and Tail

```
| ?- head([a,b,c,d]).
EXISTENCE ERROR: head([a,b,c,d]):
| ?- tail([a,b,c,d]).
EXISTENCE ERROR: tail([a,b,c,d]):
| ?- assert(p([1,2,3])).
yes
| ?- assert(p([the,cat,sat,
                   [on,the,mat]])).
yes
| ?- p([X|Y]).
X = 1,
Y = [2,3] ? ;
X = the,
Y = [cat,sat,[on,the,mat]] ? ;
no
| ?-  p([_,_,_,[_|X]]).
X = [the,mat] ? ;
no
```

Exmaple Matchings

```
| ?- [X,Y,Z] = [john,likes,fish].
X = john,
Y = likes,
Z = fish ? ;
| ?- [cat] = [X|Y].
X = cat,
Y = [] ? ;
| ?- [X,Y|Z] = [mary,likes,wine].
X = mary,
Y = likes,
Z = [wine] ? ;
```

More Examples

```
| ?- [X,Y|Z,W] = [a,b,c,d,e].
{SYNTAX ERROR: in lines 58-59}
** ] or operator expected **
?- [ X , Y | Z
** here **
, W ] = [ a , b , c , d , e ] .
| ?- [X,Y|Z] = [a,b,c,d,e].

X = a,
Y = b,
Z = [c,d,e] ? ;
```

Peculiar Examples

```
| ?- [X|Y] = "system".
X = 115,
Y = [121,115,116,101,109] ? ;
| ?- [white|Q] = [P|horse].
P = white,
Q = horse ? ;
| ?- [white|horse] = [P|Q].
P = white,
Q = horse ? ;
```

Membership in a List

`member(X,Y).`

is true when X is a member of the list Y

One of Two Conditions

X is a member of the list if X is the same as the head of the list

```
member(X,[X|_]).
```

X is a member of the list if X is a member of the tail of the list

```
member(X,[_|Y]) :-
    member(X,Y).
```

PROLOG

Not functional programing

A pattern is matched

A condition is fulfilled

Recursion

A recursive definition is used

The rules are used over and over on smaller portions of the list

Recursive Conditions

First Condition is the *boundary condition*
(A hidden boundary condition is when the list
is the empty list,
which fails)

Second Condition is the *recursive case*

Does the process ever stop?

Yes
In each recursion the list that is being checked
is getting smaller

Until
The predicate is satisfied
or
The empty list is reached

member

```
| ?- assert((member(X,[X|_]))).
true ?
yes
| ?- assert((member(X,[_|Y]) :-
                            member(X,Y))).
true ?
Yes
| ?- member(a,[a,b,c]).
 + 1  1  Call: member(a,[a,b,c]) ?
 + 1  1  Exit: member(a,[a,b,c]) ?
yes

| ?- member(b,[a,b,c]).
 + 1  1  Call: member(b,[a,b,c]) ?
 + 2  2  Call: member(b,[b,c]) ?
 + 2  2  Exit: member(b,[b,c]) ?
 + 1  1  Exit: member(b,[a,b,c]) ?
yes
```

member Failure

```
| ?- member(d,[a,b,c]).
 + 1  1  Call: member(d,[a,b,c]) ?
 + 2  2  Call: member(d,[b,c]) ?
 + 3  3  Call: member(d,[c]) ?
 + 4  4  Call: member(d,[]) ?
 + 4  4  Fail: member(d,[]) ?
 + 3  3  Fail: member(d,[c]) ?
 + 2  2  Fail: member(d,[b,c]) ?
 + 1  1  Fail: member(d,[a,b,c]) ?
no
```

Circular Definitions

```
| ?- assert((parent(X,Y):-child(Y,X))).
true ?
yes
| ?- assert((child(A,B):-parent(B,A))).
true ?
yes
| ?- parent(tom,bill).
 + 1  1  Call: parent(tom,bill) ?
 + 2  2  Call: child(bill,tom) ?
 + 3  3  Call: parent(tom,bill) ?
 + 4  4  Call: child(bill,tom) ?
 + 5  5  Call: parent(tom,bill) ?
 + 6  6  Call: child(bill,tom) ?
 + 7  7  Call: parent(tom,bill) ?
 + 8  8  Call: child(bill,tom) ?
 + 9  9  Call: parent(tom,bill) ?
 + 10  10  Call: child(bill,tom) ?
 + 11  11  Call: parent(tom,bill) ?
 + 12  12  Call: child(bill,tom) ?
forever and ever
```

Order of Predicates

```
| ?- assert((person(X) :-
                            person(Y),
                            mother(X,Y))).
true ?
yes
| ?- assert((person(adam))).
yes
| ?- person(X).
 + 1  1  Call: person(_61) ?
 + 2  2  Call: person(_315) ?
 + 3  3  Call: person(_512) ?
 + 4  4  Call: person(_709) ?
 + 5  5  Call: person(_906) ?
 + 6  6  Call: person(_1103) ?
 + 7  7  Call: person(_1300) ?
 + 8  8  Call: person(_1497) ?
 + 9  9  Call: person(_1694) ?
 + 10  10  Call: person(_1891) ?
```

```
| ?- assert((islist([A|B]) :-
                      islist(B))).
| ?- assert((islist([]))).
| ?- islist([a,b,c]).
 + 1  1  Call: islist([a,b,c]) ?
 + 2  2  Call: islist([b,c]) ?
 + 3  3  Call: islist([c]) ?
 + 4  4  Call: islist([]) ?
 + 4  4  Exit: islist([]) ?
 + 3  3  Exit: islist([c]) ?
 + 2  2  Exit: islist([b,c]) ?
 + 1  1  Exit: islist([a,b,c]) ?
yes
| ?- islist(X).
 + 1  1  Call: islist(_61) ?
 + 2  2  Call: islist(_314) ?
 + 3  3  Call: islist(_507) ?
 + 4  4  Call: islist(_700) ?
 + 5  5  Call: islist(_893) ?
 + 6  6  Call: islist(_1086) ?
 + 7  7  Call: islist(_1279) ?
```

Other Order

```
| ?- [user].
| islist2([]).
| islist2([_|Y]) :- islist2(Y).
| {user consulted, 10 msec 368 bytes}

| ?- islist2(X).
X = [] ? ;
X = [_A] ? ;
X = [_A,_B] ? ;
X = [_A,_B,_C] ? ;
X = [_A,_B,_C,_D] ? ;
X = [_A,_B,_C,_D,_E] ? ;
X = [_A,_B,_C,_D,_E,_F] ? ;
X = [_A,_B,_C,_D,_E,_F,_G] ? ;
X = [_A,_B,_C,_D,_E,_F,_G,_H] ?
```

No Loop

```
| ?- [user].
| islist([]).
| islist([_|_]).
| {user consulted, 0 msec 400 bytes}

| ?- islist([a,b,c]).
yes
| ?- islist(X).
X = [] ? ;
X = [_A|_B] ? ;
no
```