write(‘Hello World’), nl, write(‘Let\’s Program’).

>Hello World  
 Let’s Program

loves(romeo, juliet).

this is a fact where loves is predicate, romeo and juliet are constants, they start with lowercase letter

means romeo loves juliet

loves(juliet, romeo) :- loves(romeo, juliet).

this is a rule

Juliet is gonna love romeo as long as romeo loves juliet

if you say “loves(juliet, romeo).” in gprolog, then we get yes

if you say “loves(romeo, X).” in gprolog, then we get X = juliet

Here X is variable so it starts with uppercase letter

First you need to load your code to gprolog with “[filename].”

Graphical user interface, text

Description automatically generated

Some fact definitions

listing(male). 🡪 lists the males

male(X), female(Y). 🡪 show all the combinations of male and female

Soru işareti gördüğünde “;” yaz

A picture containing diagram

Description automatically generated

runs(albert) :- happy(albert).

We are saying albert runs if he is happy

dances(alice) :- happy(alice), with\_albert(alice).

Alice dances if she is happy and she is with albert.

does\_alice\_dance :- dances(alice), write(‘When Alice is happy and with Albert she dances’).

We can write “does\_alice\_dance.” in gprolog and see yes.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

~w: variable  
~s: string

Background pattern

Description automatically generated

yani variable, carl da olabilir A diye başka bir variable da.



Tybalt stabs Mercutio with a sword.

male(\_).

say true for all the male defined but not show their name. \_ means I don’t care.

Graphical user interface, text, application, email

Description automatically generated

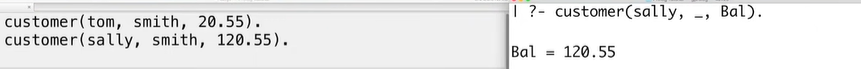
is 🡪 means equals

**STRUCTURE**

has(albert, olive). 🡪 what does this mean?

owns(albert, pet(cat, olive)). 🡪 albert owns a pet, it is a cat named olive

second one clarifies things more.



I don’t care about sally’s last name

A picture containing graphical user interface

Description automatically generated

~2f 🡪 2 digits after dot

vertical(line(point(X, Y), point(X, Y2))).

horizontal(line(point(X, Y), point(X2, Y))).

🡪 vertical(line(point(5, 10), point(5, 20))). = yes

🡪 vertical(line(point(5, 10), point(X, 20))). = X = 5

🡪 vertical(line(point(5, 10), X)). 🡪 X = point(5,\_) ---> ask for the X and Y points for a line to be vertical

**COMPARISONS**

alice = alice. 🡪 yes

‘alice’ = alice. 🡪 yes

\+ (alice = albert). 🡪 yes -----------> (How to check for not equal)

3 > 15 🡪 no

3 >= 15 🡪 no

3 =< 15 🡪 yes

W = alice 🡪 yes (This says that we can assign the value of alice to W and not that W is equal to alice)

Rand1 = Rand2. 🡪 yes

rich(money, X) = rich(Y, no\_debt). 🡪 yes (can be equal if X is assigned to no\_debt and Y is assigned to money)

**TRACE**

Using trace we can see how Prolog evaluates queries one at a time

> trace.

> mammal(human).

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

notrace. 🡪 switch off the debugger

**RECURSION**

parent(albert, bob).

parent(albert, betsy).

parent(albert, bill).

parent(alice, bob).

parent(alice, betsy).

parent(alice, bill).

parent(bob, carl).

parent(bob, charlie).

related(X, Y) :- parent(X, Y).

----> related(albert, bob). = true

related(X, Y) :-

parent(X, Z),

related(Z, Y).

----> related(albert,carl). = true

* Cycles through possible results until related returns a true

1. parent(albert, Z). = true = Z = bob, betsy, bill

2. related(Z, carl). = true when Z = bob

Graphical user interface, text, application

Description automatically generated

**MATH**

X is 2 + 2 🡪 X = 4

5+4 =:= 4+5. 🡪 yes (Check for equality between expressions)

5+4 =\= 4+5. 🡪 yes (Check for non-equality between expressions)

5 > 10 ; 10 < 100. (Checks if 1 OR the other is true)

* You can also use or operation in your rules instead of putting a comma

X is mod(7,2). 🡪 X = 1 (Modulus)

double\_digit(X,Y) :- Y is X\*2.  
% double\_digit(4,Y). = Y = 8

random(0,10,X). 🡪 (Get random value between 0 and 10)

between(0,10,X). 🡪 (Get all values between 0 and 10)  
Assign values between 0 and 10 to X

succ(2,X). 🡪 (Add 1 to 2 and assign it to X)

X is abs(-8). 🡪 (Get absolute value of -8)

X is max(10,5).

X is min(10,5).

X is round(10.56).

X is truncate(10.56).

X is floor(10.56).

X is ceiling(10.56).

X is 2\*\*3. 🡪 23

X is 11/2 🡪 X = 5.5

X is 11//2 🡪 X = 5

is\_even(X) :- Y is X//2, X =:= 2 \* Y.

Others:

sqrt, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, asinh, acosh, atanh, log, log10, exp, pi, e

**INPUT/OUTPUT**

write(‘You saw me’), nl.

You saw me

writeq(‘I show quotes’), nl.

‘I show quotes’

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**FILE**

Text

Description automatically generated

Text, letter

Description automatically generated

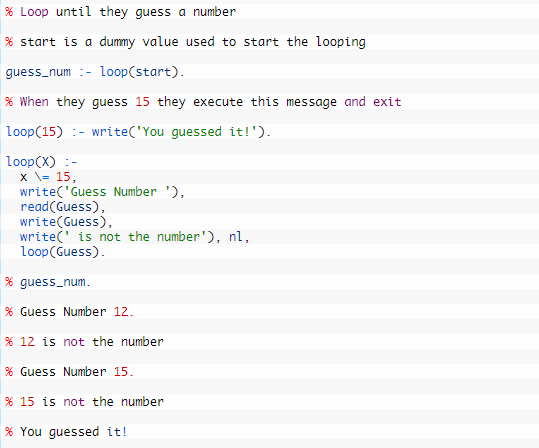
**LOOP**

count\_to\_10(10) :- write(10), nl.

count\_to\_10(X) :- write(X), nl, Y is X + 1, count\_to\_10(Y).

count\_down(Low, High) :- between(Low, High, Y), Z is High – Y, write(Z), nl.

count\_up(Low, High) :- between(Low, High, Y), Z is Y + Low, write(Z), nl.



**CHANGING THE DATABASE**

If you plan to change any predicate, you need to mark them as dynamic.

:- dynamic(father/2).

* We are saying there is father predicate that is gonna receive 2 attributes

:- dynamic(likes/2).

:- dynamic(friend/2).

:- dynamic(stabs/3).

father(lord\_montague,romeo).

father(lord\_capulet,juliet).

likes(mercutio,dancing).

likes(benvolio,dancing).

likes(romeo,dancing).

likes(romeo,juliet).

likes(juliet,romeo).

likes(juliet,dancing).

friend(romeo,mercutio).

friend(romeo,benvolio).

% friend(X, romeo) :- friend(romeo, X).

stabs(tybalt,mercutio,sword).

stabs(romeo,tybalt,sword).

We can add new clause to the database at the end of the list at the terminal:

* assertz(friend(benvolio, mercutio)).
* friend(benvolio, mercutio). 🡪 yes

Add clause at the start of the predicate list:

* asserta(friend(mercutio, benvolio)).
* friend(mercutio, benvolio). 🡪 yes

Delete a clause:

* retract(likes(mercutio,dancing)).
* likes(mercutio,dancing). 🡪 no

Delete all clauses that match:

* retractall(father(\_,\_)).
* father(lord\_montague,romeo). 🡪 no

Delete all matching clauses:

* retractall(likes(\_,dancing)).
* likes(\_,dancing). 🡪 no

**LISTS**

You can store atoms, complex terms, variables, numbers and other lists in a list

They are used to store data that has an unknown number of elements

We can add items to a list with the | (List Constructor):

* write([albert|[alice, bob]]), nl. 🡪 Adds albert to the beginning of list that has alice and bob

Get the length of a list:

* length([1,2,3], X).

We can divide a list into its head and tail with |:

* [H|T] = [a,b,c].
  + H = a
  + T = [b, c]

We can get additional values by adding more variables to the left of |:

* [X1, X2, X3, X4|T] = [a,b,c,d].
  + T = []
  + X1 = a
  + X2 = b
  + X3 = c
  + X4 = d

We can use the anonymous variable \_ when we need to reference a variable but we don’t want its value

Let’s get the 2nd value in the list:

* [\_, X2, \_, \_|T] = [a,b,c,d].
  + T = []
  + X2 = b

We can use | to access values of lists in lists:

* [\_, \_, [X|Y], \_, Z|T] = [a, b, [c, d, e], f, g, h].
  + T = [h]
  + X = c
  + Y = [d,e]
  + Z = g

Find out if a value is in a list with member:

* List1 = [a,b,c].
* member(a, List1). 🡪 yes

We could also get all members of a list with a variable:

* member(X, [a, b, c, d]).

Reverse a list:

* reverse([1,2,3,4,5], X).

Concatenate 2 lists:

* append([1,2,3], [4,5,6], X).

Write items in list on separate line:

* write\_list([]).
* write\_list([Head|Tail]) :-

write(Head), nl,

write\_list(Tail).

* % write\_list([1,2,3,4,5]). = Outputs the list

**STRINGS**

Convert a string into an Ascii character list:

* name('A random string', X).

Convert a Ascii character list into a string:

* name(X, [65,32,114,97,110,100,111,109,32,115,116,114,105,110,103]).

Append can join strings:

* join\_str(Str1, Str2, Str3) :-
  + % Convert strings into lists
  + name(Str1, StrList1),
  + name(Str2, StrList2),
  + % Combine string lists into new string list
  + append(StrList1, StrList2, StrList3),
  + % Convert list into a string
  + name(Str3, StrList3).
* join\_str(‘Another ‘, ‘Random String’, X). = X = ‘Another Random String’

get the 1st char from a string:

* name('Derek', List),
* nth0(0, List, FChar),
* put(FChar).

Get length of the string:

* atom\_length('Derek',X).

https://www.newthinktank.com/2015/08/learn-prolog-one-video/