PROLOG LAB.

Prolog;

- Yapay zeka uygulamalarında kullanılan bir **mantık programlama** dilidir.
- 1970'lerde Fransa'da icat edilmiştir; ismi ***Pro****grammation en* ***Log****ique* ifadesinden türetilmiştir [Wikizero].
- Kaynak dosyalarının uzantısı .pl olmalıdır.
- Kaynak dosyalar, *SWI Prolog* menüsünden *File > Consult…* ile yüklenebilir.
- Kaynak dosyalarda değişiklik yapıldığında *File > Reload modified files* ile değişiklikler çalıştırılabilir.

Gerçekler

- Prolog, gerçekler (*facts*), kurallar(*rules*) ve sorgular(*query*) üzerinde işleyen bir dildir.
- Gerçekler: hava soğuk, sarman bir kedidir, ali veliyi sever.
- Yukarıdaki gerçekler matematiksel mantık'a (*predicate logic*) Prolog ile şu şekilde çevrilebilir (Türkçe karakter ve büyük harf kullanmıyoruz):

```
soguk(hava).
kedi(sarman).
sever(ali, veli).
```

```
?- soguk(hava).
true.
?- kedi(sarman).
true.
?- sever(ali, veli).
true.
?- sever(ali, sarman).
false.
?- kedi(veli).
false.
```

ornek1.pl

Kurallar

- Kurallar, gerçekler arasında ilişki kurarak kendi sanal dünyamızı oluşturmamızı ve çıkarımlar yapabilmemizi sağlar. (Değişkenler büyük harfle başlamalıdır.)

```
sever(ali, veli).
sever(ali, ayse).
sever(veli, ali).

kanka(X, Y) :-
      sever(X, Y),
      sever(Y, X).

arkadas(X, Y) :-
      sever(X, Y);
      sever(Y, X).
```

```
?- sever(ali, ayse).
true.
?- sever(ayse, ali).
false.
?- arkadas(ayse, ali).
true.
?- kanka(ayse, ali).
false.
?- kanka(veli, ali).
true.
```

ornek2.pl

Sorgular

- Sorgular ile gerçekler ve kurallar ile tanımladığımız dünya hakkında bilgi edinebiliriz.

```
sever(ali, veli).
sever(ali, ayse).
sever(veli, ali).

kanka(X, Y) :-
        sever(X, Y),
        sever(Y, X).

arkadas(X, Y) :-
        sever(X, Y);
        sever(Y, X).
```

```
?- sever(ali, veli).
true.
?- sever(ali, X).
X = veli ;
X = ayse.
?- kanka(veli, X).
X = ali.
?- kanka(ayse, X).
false.
?- kanka(Birinci_kisi, Ikinci_kisi).
Birinci_kisi = ali,
Ikinci_kisi = veli ;
Birinci_kisi = veli,
Ikinci_kisi = ali.
```

Önemsiz Değişkenler

- Prolog'da önemsiz değişkenleri (don't care variable) _ ile temsil edilir.

```
yer(kedi, hamsi, balik).
yer(kedi, istavrit, balik).
yer(kopek, kuru, mama).
yer(kopek, yas, mama).
yer(kopek, istavrit, balik).
```

```
?- yer(X, Y, balik).
X = kedi,
Y = hamsi ;
X = kedi,
Y = istavrit ;
X = kopek,
Y = istavrit.

?- yer(X, _, balik).
X = kedi ;
X = kedi ;
X = kopek.

?- yer(X, _, balik), yer(X, _, mama).
X = kopek ;
X = kopek.
```

Aritmetik Operatörler

- Prolog'da bir değişkene sayısal değer ***is*** ile atanır.

```
?- X is 3.
X = 3.

?- X is 3+4.
X = 7.

?- X is 3*4.
X = 12.

?- X is 3/4.
X = 0.75.

?- X is 3//4.
X = 0.
```

```
?- 5 >= 3.
true.

?- 3 =< 1.
false.

?- 3+6 =:= 8+1.
true.

?- 3+6 =\= 2+1.
true.

?- X is 12 mod 5.
X = 2.
```

Örnek 1

Aşağıdaki tablodaki verilere göre istenilen şehrin istenilen yıldaki sıcaklığını Fahrenheit cinsinden bulan Prolog programını yazınız.

(F = 9/5 * C + 32)

Şehir	Yıl	Ort. Sıcaklık (C)
Ankara	2017	23
Ankara	2018	24
İstanbul	2017	26
İstanbul	2018	27
Bolu	2017	21
Bolu	2018	22

Örnek çıktı:

```
?- fahrenheit(bolu, 2017).
bolu sehrinin 2017 yilindaki ortalama sicakligi 69 Fahrenheittir.
```

Örnek 1 Çözüm

```
celcius(ankara, 2017, 23).
celcius(ankara, 2018, 24).
celcius(istanbul, 2017, 26).
celcius(istanbul, 2018, 27).
celcius(bolu, 2017, 21).
celcius(bolu, 2018, 22).

fahrenheit(Sehir, Yil) :-
        celcius(Sehir, Yil, Celc),
        F is (Celc*1.8+32),
        write(Sehir), write(' sehrinin '),
        write(Yil), write(' yilindaki ortalama sicakligi '),
        write(F), write(' Fahrenheittir.'), nl.
```

ornek3.pl

Listeler

- Prolog'da listeler [e_0, e_1, e_2, …, e_n] şeklinde gösterilir. Liste elemanları hetorojen olabilir.
- Listenin elemanlarına [Baş|Gerisi], [B1, B2|G] gibi yöntemler ile erişilebilir.

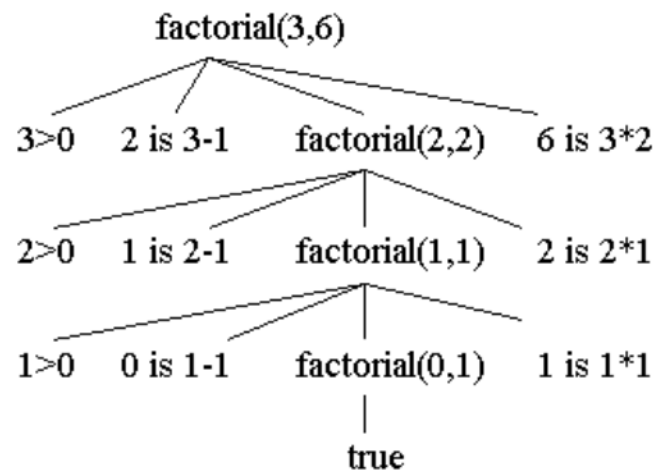
```
?- [B|G] = [a, b, c, d, [e, f], 7].
B = a,
G = [b, c, d, [e, f], 7].

?- [B0,B1|G] = [a, b, c, d, [e, f], 7].
B0 = a,
B1 = b,
G = [c, d, [e, f], 7].

?- [_, _, _, [B|_]|_] = [a, icer(kedi, sut), 5, [x, y, z], ali, veli].
B = x.
```

Özyineleme (Recursion)

- Prolog'da *for, while* gibi döngü belirteçleri bulunmadığı için bazı problemler özyineleme (recursion) ile çözülmelidir.

```
/* factorial(0, 1). */

factorial(0, F) :-
        F is 1.

factorial(N, F) :-
        N > 0,
        Nr is N-1,
        factorial(Nr, Fr),
        F is N * Fr.
```

```
?- trace.
true.

[trace]  ?- factorial(3, N).
   Call: (8) factorial(3, _5034) ? creep
   Call: (9) 3>0 ? creep
   Exit: (9) 3>0 ? creep
   Call: (9) _5274 is 3+ -1 ? creep
   Exit: (9) 2 is 3+ -1 ? creep
   Call: (9) factorial(2, _5276) ? creep
   ...
   Exit: (8) factorial(3, 6) ? creep
N = 6 .

[trace]  ?- notrace.
true.
```



https://www.cpp.edu/~jrfisher/www/prolog_tutorial/2_2.html

Özyineleme (Recursion) devamı / 2

- Önceki örnekteki faktöryel hesaplama fonksiyonunu kullanıcıdan N sayısını alacak şekilde tekrar yazınız.

```
factorial(0, 1).

factorial(N, F) :-
        N > 0,
        Nr is N-1,
        factorial(Nr, Fr),
        F is N * Fr.

program :-
        write('Bir sayi giriniz: '),
        read(X),
        nl,
        factorial(X, Sonuc),
        write(X), write('! = '), write(Sonuc), nl.
```

```
?- program.
Bir sayi giriniz: 5.

5! = 120
true .

ornek4.pl
```

Örnek

Girilen bir elemanı girilen bir liste içerisinde arayan Prolog programını yazınız.

Örnek çıktı:

```
?- elemani(ali, [1, [a, b], ali, veli]).
true .

?- elemani(5, [1, 3, 4]).
false.
```

Örnek Çözüm

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

elemani(X, [Y|G]) :-
        X \= Y,
        elemani(X, G).
```

```
?- trace.
true.

[trace]  ?- elemani(ali, [1, [a, b], ali, veli]).
   Call: (8) elemani(ali, [1, [a, b], ali, veli]) ?
creep
   Call: (9) ali\=1 ? creep
   Exit: (9) ali\=1 ? creep
   Call: (9) elemani(ali, [[a, b], ali, veli]) ?
   ...
creep
   Exit: (8) elemani(ali, [1, [a, b], ali, veli]) ?
creep
true .

[trace]  ?- notrace.
true.
```

Ornek5.pl

Örnek

Girilen bir elemanı girilen bir - iç içe - liste içerisinde arayan Prolog programını yazınız.

Örnek çıktı:

```
?- elemani(ali, [1, [a, b], [[[ayse, ali]]], veli]).
true .

?- elemani(ali, [1, [a, b], ali, veli]).
true .
```

Ornek6.pl

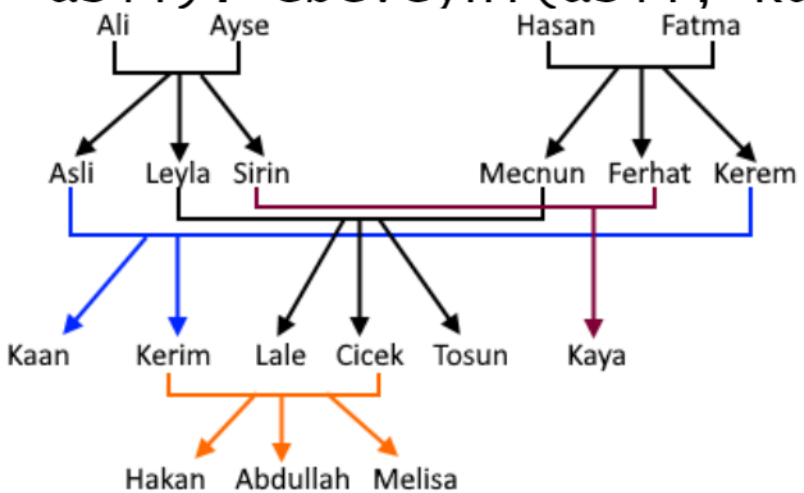
Örnek Çözüm

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

elemani(X, [Y|_]) :-
        X \= Y,
        elemani(X, Y).

elemani(X, [Y|G]) :-
        X \= Y,
        elemani(X, G).
```

```
?- trace.
true.

[trace]  ?- elemani(ali, [1, [a, b], [[[ayse, ali]]], veli]).
   Call: (8) elemani(ali, [1, [a, b], [[[ayse, ali]]], veli]) ? creep
   Call: (9) ali\=1 ? creep
   Exit: (9) ali\=1 ? creep
   Call: (9) elemani(ali, 1) ? creep
   Fail: (9) elemani(ali, 1) ? creep
   Redo: (8) elemani(ali, [1, [a, b], [[[ayse, ali]]], veli]) ? creep
   Call: (9) ali\=1 ? creep
   ...
   Call: (14) elemani(ali, [ali]) ? creep
   Exit: (14) elemani(ali, [ali]) ? creep
   Exit: (13) elemani(ali, [ayse, ali]) ? creep
   Exit: (12) elemani(ali, [[ayse, ali]]) ? creep
   Exit: (11) elemani(ali, [[[ayse, ali]]]) ? creep
   Exit: (10) elemani(ali, [[[[ayse, ali]]], veli]) ? creep
   Exit: (9) elemani(ali, [[a, b], [[[ayse, ali]]], veli]) ? creep
   Exit: (8) elemani(ali, [1, [a, b], [[[ayse, ali]]], veli]) ? creep
true .
```

Örnek

- Aşağıda akrabalık ilişkileri gösterilen kişileri tanımlayınız ve `atasi_mi`, `kardesi_mi`, `soyundan_mi`, `cocugu_var_mi`, `teyzesi_mi`, `amcasi_mi`, `dayisi_mi`, `halasi_mi`, `yengesi_mi` fonksiyonlarini oluşturunuz.
- Gerçekleri (facts) tanımlarken sadece *ebeveyni(x, y) ile erkek(x), kadin(y)* kullanabilirsiniz. Yani *dedesi(a, b), kardeşi(y, x)* veya *buyuk_torunu(k, l)* gibi gerçekler kullanmayınız. Örn: `ebeveyni(ali, asli). ebeveyni(asli, kaan), kadin(asli), erkek(kaan).`



Örnek / 2

Örnek çıktılar aşağıda gösterilmiştir.

```
?- atasi_mi(hasan, mecnun).
true .
?- atasi_mi(ayse, hakan).
true .
?- atasi_mi(hasan, melisa).
true .
?- atasi_mi(mecnun, abdullah).
true .
?- atasi_mi(ferhat, hakan).
false.
?- atasi_mi(leyla, ali).
false.
?- atasi_mi(leyla, leyla).
false.
```

```
?- soyundan_mi(ali, ali).
false.
?- soyundan_mi(asli, ali).
true .
?- soyundan_mi(abdullah, ayse).
true .
?- soyundan_mi(abdullah, ferhat).
false.
```

```
?- kardesi_mi(ali, ayse).
false.
?- kardesi_mi(mecnun, kerem).
true .
?- kardesi_mi(mecnun, mecnun).
false.
?- kardesi_mi(hakan, abdullah).
true .
```

```
?- cocugu_var_mi(ali).
true .
?- cocugu_var_mi(kaya).
false.
```