UNIFICATION

Prolog program

```
tu(X,Y) :- X=Y,nl,write(X).

% unification in Prolog is obtained through the infix operator =
% attempt to unify X and Y, feed a new line, write the result of the unification or return failure
```

Trial runs

```
?- tu([number:sg],Feature).
[number:sg]
Feature = [number:sg].

?- tu([number:sg],[number:pl]).
false.

?- tu([number:Number],[number:pl]).
[number:pl]
Number = pl.

?- tu([number:Number,gender:G],[number:pl,gender:Gender]).
[number:pl,gender:_G560]
Number = pl,
G = Gender.

?- tu([number:Number,gender:masc],[number:pl,gender:Gender]).
[number:pl,gender:masc]
Number = pl,
Gender = masc.

?- tu(head:[number:Number,gender:masc],head:[number:pl,gender:Gender]).
head:[number:pl,gender:masc]
Number = pl,
Gender = masc.

?- tu(head:[number:Number,gender:masc],head:[number:pl,gender:fem]).
false.

?- tu(head:[number:Number,gender:masc],Result).
head:[number:_G488,gender:masc]
Result = head:[number:Number, gender:masc].
```

Constraint : checking that first arg 'matches' second arg

```
?- constraint([gender:masc],[number:sg, gender:masc]).
true.

?- constraint([gender:masc],[number:sg, gender:Gender]).
Gender = masc.

?- constraint([gender:masc],[head:[number:sg, gender:Gender]]).
false.

?- constraint([head:[gender:masc]],[head:[number:sg, gender:Gender]]).
Gender = masc.

?- constraint([head:[gender:masc]],[head:[number:sg, gender:fem]]).
false.

?- constraint([head:[gender:G]],[head:[number:sg, gender:Gender]]).
G = Gender.

?- constraint([head:[gender:G]],[head:[number:sg, gender:G]]).
true.

?- constraint([head:[gender:fem]],[head:[number:sg, gender:G,function:subject]]).
G = fem.

?- constraint([head:Head],[head:[number:sg, gender:fem,function:subject]]).
Head = [number:sg, gender:fem, function:subject].

?- constraint(Feature,[head:[number:sg, gender:fem,function:subject]]).
Feature = [head:[number:sg, gender:fem, function:subject]].
```

List unification

```
% a list is an ordered sequence of elements included in square brackets
% the | operator isolates any number of elements at the head of a list
% the tail of a list is always a list
% the empty list is written []


?- tu(List,[a,b|Tail]).
[a,b|_G347]
List = [a, b|Tail].

?- tu([Prem,b,c,d],[a,b|Tail]).
[a,b,c,d]
Prem = a,
Tail = [c, d].

?- tu([Prem|Queue],[a,b|Tail]).
[a,b|_G398]
Prem = a,
Queue = [b|Tail].

?- tu([],[]).
[]
true.

?- tu([a|Tail],[a]).
[a]
Tail = [].

?- tu([a|Tail],[b,a,c|Queue]).
false.

?- tu([a,b,c|Tail],[b,a,c|Queue]).
false.

?- tu([b,a,c,d,e|Tail],[b,a,c|Queue]).
[b,a,c,d,e|_G446]
Queue = [d, e|Tail].

?- tu([a,[b,c],d],[Prem,Sec|Third]).
[a,[b,c],d]
Prem = a,
Sec = [b, c],
Third = [d].
```

```
?- tu([a,[b,c],d],[Prem,[A,D]|Third]).
[a,[b,c],d]
Prem = a,
A = b,
D = c,
Third = [d].

?- tu(['Albert','Bernard', nom(albert,durant), adresse([inconnue])],[Prem,Sec,Name,Indirizzo]).
[Albert,Bernard,nom(albert,durant),adresse([inconnue])]
Prem = 'Albert',
Sec = 'Bernard',
Name = nom(albert, durant),
Indirizzo = adresse([inconnue]).

?- tu([gender:G,number:sg],[Gender,number:Number]).
[gender:_G492,number:sg]
Gender = gender:G,
Number = sg.

?- tu([gender:masc,Number],[Gender,number:N]).
[gender:masc,number:_G474]
Number = number:N,
Gender = gender:masc.

?- tu([gender:masc,Number],[Gender,number:pl]).
[gender:masc,number:pl]
Number = number:pl,
Gender = gender:masc.
```