Predicates as procedures, and arguments as i/o

% increment(+X,-Y)
increment(X,Y) :- Y is X+1.
Predicates as procedures, and arguments as i/o

% increment(+X,-Y)
increment(X,Y) :- Y is X+1.

% incr(-X,+Y)
incr(X,Y) :- X is Y-1.
Predicates as procedures, and arguments as i/o

% increment(+X,-Y)
increment(X,Y) :- Y is X+1.

% incr(-X,+Y)
incr(X,Y) :- X is Y-1.

% incr2(+X,+Y)
incr2(X,Y) :- X =:= Y-1.
Predicates as procedures, and arguments as i/o

% increment(+X,-Y)
increment(X,Y) :- Y is X+1.

% incr(-X,+Y)
incr(X,Y) :- X is Y-1.

% incr2(+X,+Y)
incr2(X,Y) :- X =:= Y-1.

From SWI Prolog documentation:
\[ -\text{Number is } +\text{Expr} \]
True when \text{Number} is the value to which \text{Expr} evaluates.
Predicates as procedures, and arguments as i/o

% increment(+X,-Y)
increment(X,Y) :- Y is X+1.

% incr(-X,+Y)
incr(X,Y) :- X is Y-1.

% incr2(+X,+Y)
incr2(X,Y) :- X =:= Y-1.

From SWI Prolog documentation:

- \textit{Number} is \textit{+Expr}
  
  True when \textit{Number} is the value to which \textit{Expr} evaluates.

- \textit{+Expr1} =:= \textit{+Expr2}

  True if expression \textit{Expr1} evaluates to a number equal to \textit{Expr2}. 
Mode indicators

+ input (known)
− output (unknown)

From SWI Prolog documentation

An argument mode indicator gives information about the intended direction in which information carried by a predicate argument is supposed to flow. Mode indicators (and types) are not a formal part of the Prolog language but help in explaining intended semantics to the programmer. There is no complete agreement on argument mode indicators in the Prolog community.
Mode indicators

+ input (known)
− output (unknown)

From SWI Prolog documentation

An argument mode indicator gives information about the intended direction in which information carried by a predicate argument is supposed to flow. Mode indicators (and types) are not a formal part of the Prolog language but help in explaining intended semantics to the programmer. There is no complete agreement on argument mode indicators in the Prolog community.

? uncommitted (don’t care \(\approx\) unknown)

% successor(\(?X,?Y)\)
successor(X, succ(X)).
Mode indicators

+ input (known)
– output (unknown)

From SWI Prolog documentation

An argument mode indicator gives information about the intended direction in which information carried by a predicate argument is supposed to flow. Mode indicators (and types) are not a formal part of the Prolog language but help in explaining intended semantics to the programmer. There is no complete agreement on argument mode indicators in the Prolog community.

? uncommitted (don’t care ≈ unknown)

% successor(?X,?Y)
successor(X,succ(X)). % :- numeral(X).
% numeral(?X)
n numeral(0).
n numeral(succ(X)) :- numeral(X).
Reversibility with ?

\(?Term1 = ?Term2\)

Unify \(Term1\) with \(Term2\). True if the unification succeeds.
Reversibility with \( \? \)

\(?Term1 = ?Term2\)

Unify \(Term1\) with \(Term2\). True if the unification succeeds.

member(\(?Elem,?List\))

True if \(Elem\) is a member of \(List\).
Reversibility with \(?\)

\(?\text{Term1} = \)?\text{Term2}

Unify \text{Term1} with \text{Term2}. True if the unification succeeds.

\text{member(}\text{?Elem,?List)}

True if \text{Elem} is a member of \text{List}.

\text{- member(1,}\text{[1]}\).

\text{true.}
Reversibility with ?

\(?Term1 = ?Term2\)

Unify \(Term1\) with \(Term2\). True if the unification succeeds.

\(\text{member}(?\text{Elem}, ?\text{List})\)

True if \(Elem\) is a member of \(List\).

\(?- \text{member}(1, [1]).\)

true.

\(?- \text{member}(X, [1]).\)

\(X = 1\).
Reversibility with ?

\(?Term1 = ?Term2\)

Unify \(Term1\) with \(Term2\). True if the unification succeeds.

member(?Elem,?List)

True if \(Elem\) is a member of \(List\).

?- member(1,[1]).
true.

?- member(X,[1]).
X = 1  .

?- member(1,List).
List = [1|_] ;
List = [_,1|_] ;
...

Reversibility with ?

\(?Term1 = ?Term2\)

Unify \(Term1\) with \(Term2\). True if the unification succeeds.

member(?Elem,?List)

True if \(Elem\) is a member of \(List\).

?- member(1,[1]).
  true.

?- member(X,[1]).
  X = 1 .

?- member(1,List).
  List = [1|_] ;
  List = [_,1|_] ;
  ...

?- member(X,List).
  List = [X|_] ;
  List = [_,X|_] ;
  ...
Two more mode indicators

@ argument will *not* be further instantiated

@Term1 == @Term2
   True if Term1 is equivalent to Term2.

var(@Term)
   True if Term currently is a free variable.
Two more mode indicators

@ argument will not be further instantiated

@Term₁ == @Term₂
  True if Term₁ is equivalent to Term₂.

var(@Term)
  True if Term currently is a free variable.

: meta-argument that can be called as goal

\+ :Goal
  True Goal cannot be proven

call(:Goal₁)
  Call Goal.
### On swipl

if(A,B,C) :- (A,! , B) ; C.

neg(A) :- if(A,fail,true).

?- listing(if).
?- if(0=0,X=1,X=2).
X = 1.
?- if(0=1,X=1,X=2).
X = 2.
?- 0=0 -> X=0.
X = 0.
?- 0=1 -> X=1.
false.
On swipl

if(A,B,C) :- (A,! ,B) ; C.

neg(A) :- if(A,fail,true).

?- listing(if).
if(A, B, C) :-
    ( call(A), !, call(B) ; call(C) ).
true.
On swipl

if(A,B,C) :- (A,!,,B) ; C.
	neg(A) :- if(A,fail,true).

?- listing(if).
if(A, B, C) :-
    ( call(A), !, call(B) ; call(C) ).
true.

?- if(0=0,X=1,X=2).
X = 1.

?- if(0=1,X=1,X=2).
X = 2.
On swipl

if(A, B, C) :- (A, !, B) ; C.

neg(A) :- if(A, fail, true).

?- listing(if).
if(A, B, C) :-
    ( call(A), !, call(B) ; call(C) ).
true.

?- if(0 = 0, X = 1, X = 2).
X = 1.

?- if(0 = 1, X = 1, X = 2).
X = 2.

?- 0 = 0 -> X = 0.
X = 0.

?- 0 = 1 -> X = 1.
false.