Prolog Cheat Sheet

A concise reference for Prolog syntax, predicates, and common programming patterns.

Basic Syntax and Data Types

Facts and Rules

Facts:	Declare relationships between objects. `parent(john, mary).` (John is a parent of Mary)
Rules:	Define conditional relationships. `ancestor(X, Y) :- parent(X, Y).` (X is an ancestor of Y if X is a parent of Y) `ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).` (X is an ancestor of Y if X is a parent of Z and Z is an ancestor of Y)
Queries:	Ask questions about the relationships. `?- parent(john, mary).` (Is John a parent of Mary?) `?- ancestor(john, Y).` (Who are John's descendants?)

Data Types

Atoms:	Constants, starting with a lowercase letter. Examples: `john`, `mary`, `cat`
Numbers:	Integers and floating-point numbers. Examples: `1`, `3.14`, `-5`
Variables:	Start with an uppercase letter or underscore. Examples: `X`, `Y`, `_Result`
Structures:	Complex terms, combining a functor (name) and arguments. Example: `book(title, author)`
Lists:	Ordered collections of terms. Example: `[1, 2, 3]`, `[a, b, c]` `[Head | Tail]` - Represents a list with Head as the first element and Tail as the rest of the list.

Operators

`:-`	Rule definition (if).
`,`	Conjunction (and).
`;`	Disjunction (or).
`=`	Unification (attempt to make terms identical).
`\=`	Not unifiable.

List Manipulation

Basic List Operations

Lists are a fundamental data structure in Prolog. They are enclosed in square brackets `[]` and elements are separated by commas.

`[Head | Tail]` notation is used to represent a list, where `Head` is the first element and `Tail` is the rest of the list.

Predicates for List Manipulation

`member(X, List)`	Succeeds if `X` is an element of `List`.
	`?- member(b, [a, b, c]).`
	`true.`

`append(List1, List2, List3)`	Succeeds if `List3` is the result of appending `List1` and `List2`.
	`?- append([a, b], [c, d], X).`
	`X = [a, b, c, d].`

`length(List, Length)`	Succeeds if `Length` is the length of `List`.
	`?- length([a, b, c], X).`
	`X = 3.`

`reverse(List, ReversedList)`	Succeeds if `ReversedList` is the reverse of `List`.
	`?- reverse([a, b, c], X).`
	`X = [c, b, a].`

Example: Defining `member`

```
member(X, [X | _]).  % X is a member if it's the head.
member(X, [_ | Tail]) :- member(X, Tail).  % Otherwise, check the tail.
```

Arithmetic Operations

Basic Arithmetic

`is`	Used to evaluate arithmetic expressions. `X is Expression` assigns the result of `Expression` to `X`. **Note:** The right-hand side must be fully evaluable.
+, -, *, /	Standard arithmetic operators.
`mod`	Modulo operator (remainder of division). `X is 7 mod 2.` (X will be 1)

Comparison Operators

`=:=`	Arithmetic equality (values are equal).
`=\=`	Arithmetic inequality (values are not equal).
`<, >, =<, >=`	Less than, greater than, less than or equal to, greater than or equal to.

Example: Factorial

```
factorial(0, 1).  % Base case: factorial
of 0 is 1.
factorial(N, F) :-   % Recursive case:
    N > 0,          % N must be greater
than 0.
    N1 is N - 1,    % Calculate N - 1.
    factorial(N1, F1), % Calculate
factorial of N - 1.
    F is N * F1.     % F is N *
factorial(N-1).
```

Control Flow and Logic

Cut (`!`)

The cut (`!`) is a goal that always succeeds, but with a side effect: it commits Prolog to the choices made so far in the current rule.
It prevents backtracking.

Use with caution, as it can make programs harder to understand and debug.

Negation as Failure

`\+ Goal`	Succeeds if `Goal` fails. This is *negation as failure*: Prolog assumes something is false if it cannot prove it to be true.
Example:	`different(X, Y) :- \+ X = Y.` `different(a, b).` would succeed, while `different(a, a).` would fail.

Conditional Predicates

Prolog doesn’t have explicit `if-then-else` statements like imperative languages.
Instead, conditional logic is achieved through multiple rules and the use of cuts.

Example:

```
max(X, Y, X) :- X >= Y, !.
max(X, Y, Y) :- Y > X.
```

If `X >= Y`, the first rule succeeds (and the cut prevents backtracking to the second rule). Otherwise, the second rule is tried.