Prolog is *declarative*, not *imperative*. You describe logic, not a flow of control.

Facts, rules, and queries.

It compiles facts + rules (together called a *knowledge base*) to support for efficient querying.

If a symbol begins with a lowercase character it's an atom - a fixed value.
If it begins with an uppercase character it's a variable.

```
['relative/path/file.pl']
```
or
```
['/absolute/path/to/file.pl']
```

`yes` means "yes I can prove that" or "yes I know that to be true".

`no` means "no I can't prove that" or "no I don't know that to be true".

Use the `\+` predicate, `.e.g,`

```
somePred(X, Y) :- \+ (X = Y), someOtherPred(X, Y)
```

- Type a fact and see if Prolog can prove it.
- Type a fact with a variable (conventionally named `What`) to see if Prolog can find bindings that satisfy the fact. Use semicolons (`;`) to ask for another binding, or `a` to get all of them.

$=$ is **not** assignment, it is *unification*, the attempt to make both sides logically the same.

Only one of the clauses must be satisfied.

Every clause must be satisfied.

Make your recursive subgoal the last subgoal in a rule.

```
?- [one, two, three, four]= [Head | Tail].
Head = one,
Tail = [two, three, four].
```

One which has constraints that are easily expressed but difficult to satisfy. Since Prolog figures out the satisfying bindings, Prolog is doing the hard part for you.

- Games
- Semantic web
- AI and NLP
- Scheduling

- It's not a general purpose language.
- It may not scale due to inefficient DFS approach to unification, and heavy use of recursion.