What is backwards reasoning?

Why does Prolog use it?

Backwards reasoning is from (potential) conclusions to facts instead of from facts to conclusions.
Prolog uses it because the space of possible conclusions grows too quickly in the number of premises.

How can you load a Prolog database for interactive use?

You can use the `consult` and `reconsult` predicates, or use bracket notation.
Bracket notation allows re-consultation of several files at once:

```
['file1.pl', file2.pl']
```

What is the closed world assumption?

A Prolog database knows everything it needs to know.

What is a structure?

A name followed by zero or more arguments. Parens are omitted if there are no arguments.

What is a base clause?

A structure terminated by a period. It represents a simple fact.

What is a nonbase clause?

A structure followed by a turnstile and a list of structures separated by commas. It represents a rule.

What is a predicate?

A **collection** of clauses with the same *functor* (name) and arity.
This is similar to the collection of overloaded versions of a function in an imperative language.

What is a program?

A collection of predicates, in any order.

How can you include special characters and spaces in an atom, or begin it with a capital letter?

Use single quotes, which does **not** make a string.

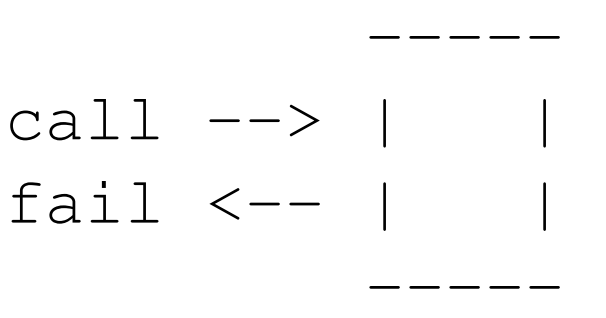
What are double quotes for?

They indicate a list of ASCII values.

How can you include escape characters in a quoted atom?

Use double quotes or an escaped single quote to use a single quote.
Other escapes use backslash as in other languages.

What are the four ports of a structure?



`exit` ports connect to `call` ports.
`fail` ports connect to `redo` ports.

Prolog's logic is *non-monotonic*. What does that mean?

Facts can be added at any time using the `assert` predicate.
Facts can be removed at any time using the `retract` predicate.
Such rules are *dynamic*.

Why might you need to double the parens of functor?

To force a rule to be interpreted as a single argument, since rules contain commas.

```
assert((loves(chuck, X) :- female(X), rich(X))).
```

What are the limitations of backtracking.

- Output can't be undone.
- `assert` and `retract` can't be undone either.

How do you write to stdout?

`write` predicate outputs its single argument to stdout.
`nl` writes a newline.

How can you view the available facts and rules?

How can you view the available structures for a predicate?

```
listing(predicate)

listing
```

What does a single underscore (_) do?

It's an anonymous variable and can represent any term.

What's the difference between cut (`!`) and `fail`?

`fail` doesn't force other the entire predicate to fail. Other clauses will be tried.

Using a cut creates a commit point, preventing backtracking past the commit point and preventing attempts on other clauses.

What happens if you combine cut and fail?

The predicate as a whole fails.

Why might you start a variable with an underscore?

To tell Prolog you are only going to use it once, but don't wish to use the anonymous variable.

What is the scope of a variable?

For onymous variables, the single clause in which it appears.

What order are clauses of a predicate tried in?

The same order in which they were defined.

What can you use in place of `if`?

What can you use in place of loops?

- Predicates with multiple clauses that have "tests" in them.
- Recursion.

How can you get around the lack of functions with no return values?

Use a fail loop.

```
my_func(X) :-
    Some(),
    Imperative(),
    Calls(),
    fail.
my_func(_)
```

It's generally bad style to do this.

How can you get around the lack of functions with return values?

A parameter (conventionally the final one) of a parameter list can be used for output.

```
?- assert((first([Head | Tail], X) :- X = Head)).
true.
?- first([1,2,3], X).
X = 1.
```

How can you work around the lack of assignable state?

You can store state in the database as facts.

```
bump_count :-
    retract(count(X)),
    Y is X + 1,
    assert(count(Y)).
```

It's generally bad style to do this.

What is the `=..` operator?

The "univ" operator. It converts between structures and lists.

```
loves(chuck, X) =.. [loves, chuck, X]
```

What predicate describes list membership?

What predicate describes combining lists?

```
?- member(1, [1, 2]).
true .

?- append([1], [2], [1, 2]).
true.
```

How can you force a predicate to succeed?

Make the final case a "dummy" with no body that uses anonymous variables for all the parameters.

What should you do if you don't care if a call succeeds and don't want failure to cause backtracking?

```
wrapper :-
    potentially_failing_call.
wrapper.
```

Now any call to wrapper will succeed.

What are `asserta` and `assertz` for?

They are like `assert`, but `asserta` guarantees the added clause will come before any clauses with the same functor. Likewise `assertz` guarantees the added clause will be the last case in its predicate.

What does `abolish` do?

It removes ``all` clauses of the predicate with the given functor and arity.

```
abolish(somePred, arity).
```

What's the catch with arithmetic in Prolog?

`+`, `-`, `*`, `/`, and `mod` have their normal meanings, but *only when evaluated*.

They may not be evaluated when you want, leading to strange outcomes like:

```
?- 2 + 2 = 4.
false.
```

What is the difference between static and dynamic clauses in SWI Prolog?

Static clauses are the default and cannot be later modified using `assert/retract`.
Marking clauses dynamic (before they are defined) allows you to change the definition during program execution.

```
:- dynamic somePredArityTwo/2, somePredArityOne/1.
```

To actually force arithmetic to be performed you can use `is` or comparison operators like `=:=`, `=/=`, `>`, `>=`, `<`, `<=`.