Prologue to Prolog

Make Computers
Work for *You*.

?- THE CLAUSE.

See a video of this presentation at: https://share.descript.com/view/snZvbV85LdN

Why Learn Prolog?

- Learn a new programming paradigm
- Discover what's possible
- Expand your problem-solving cognition

Overview

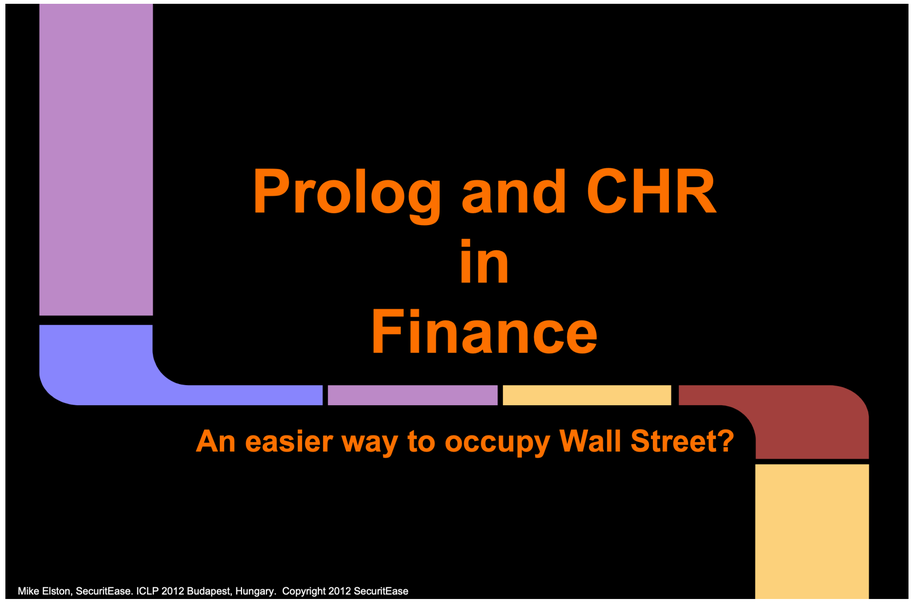
1. What can Prolog do for you?
2. A Brief Landscape of Prolog
3. Prolog Programming 101

1. What Can Prolog do for You?

“SICStus [Prolog]... handles nearly a third of all airline tickets in the world.”

https://web.archive.org/web/20190203010151/https://www.sics.se/projects/sicstus-prolog-leading-prolog-technology

“Reason *in* the application and *about* the application”



- SecuritEase, the dominant stock broking system in New Zealand
- Multi-million dollar system
- 1.5m lines of Prolog

https://dtai.cs.kuleuven.be/CHR/files/Elston_SecuritEase.pdf

- Datalog is a subset of Prolog
- TerminusDB is “a toolkit for building collaborative applications”
- Written in Prolog and Rust

“Datalog represents a stepping-stone from relational languages such as SQL

to more fully-featured programming languages

while retaining the declarative, robust, pervasive, and resilient properties of query languages.

”

https://terminusdb.com/docs/index/terminusx-db/explanations/datalog

Prolog Use Cases

WEB APPS

EXPERT SYSTEMS

CRYPTOGRAPHY

DATA ANALYTICS

RAPID PROTOTYPING

TYPE SYSTEMS

DECLARATIVE DSL'S

PARSING

2. A Brief Prolog Landscape

2. A Brief Prolog Landscape

Prolog is over 50 years old this year!

2. A Brief Prolog Landscape

One ISO standard

SWI Prolog
(C)

Tau Prolog
(JavaScript)

ISO

(Golang)

And many more!

Scryer Prolog (Rust)

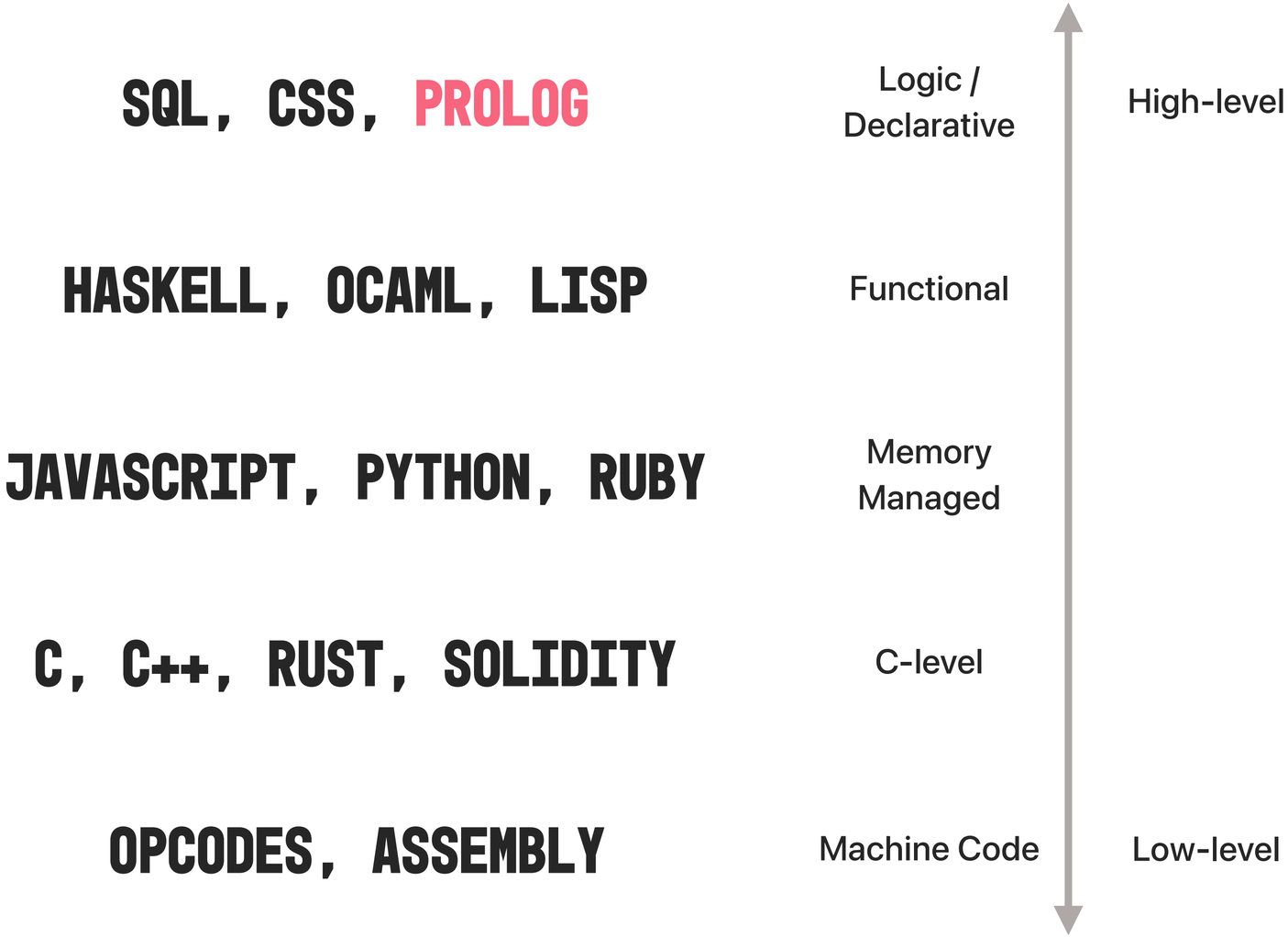
Many Implementations

3. Prolog Programming 101

3. Prolog Programming 101

Prolog is a logic programming language.

3. Prolog Programming 101



3. Prolog Programming 101

What we will cover:

- 101: The anatomy of a predicate
- 102: Predicates as data
- 103: Querying predicates

Prolog Programming 101

Logic programming is a **new** (to you) paradigm

Prolog Programming 101

OLD WAY

~~Functions~~

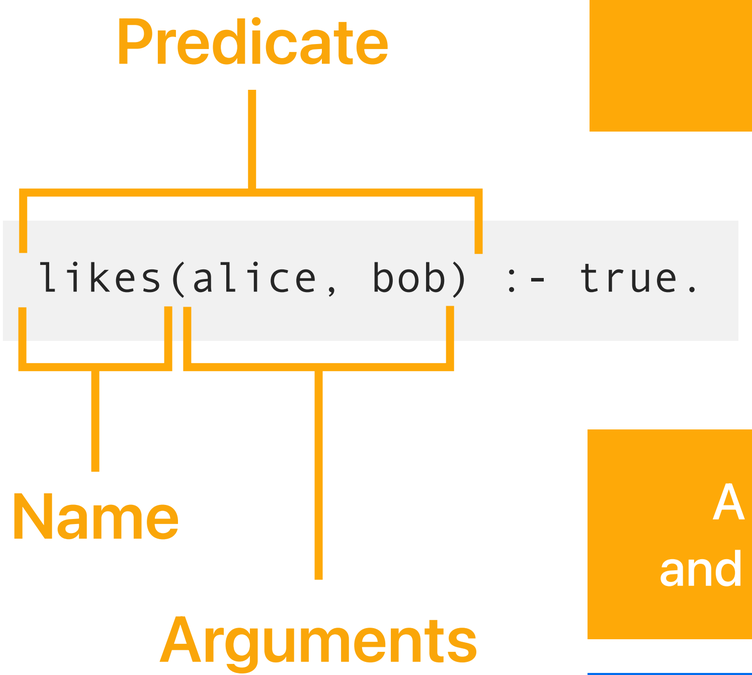
NEW WAY

Predicates

Prolog Programming 101

NEW WAY

Predicates



Here we have
a “likes” predicate.

A predicate has a name,
and zero or more arguments.

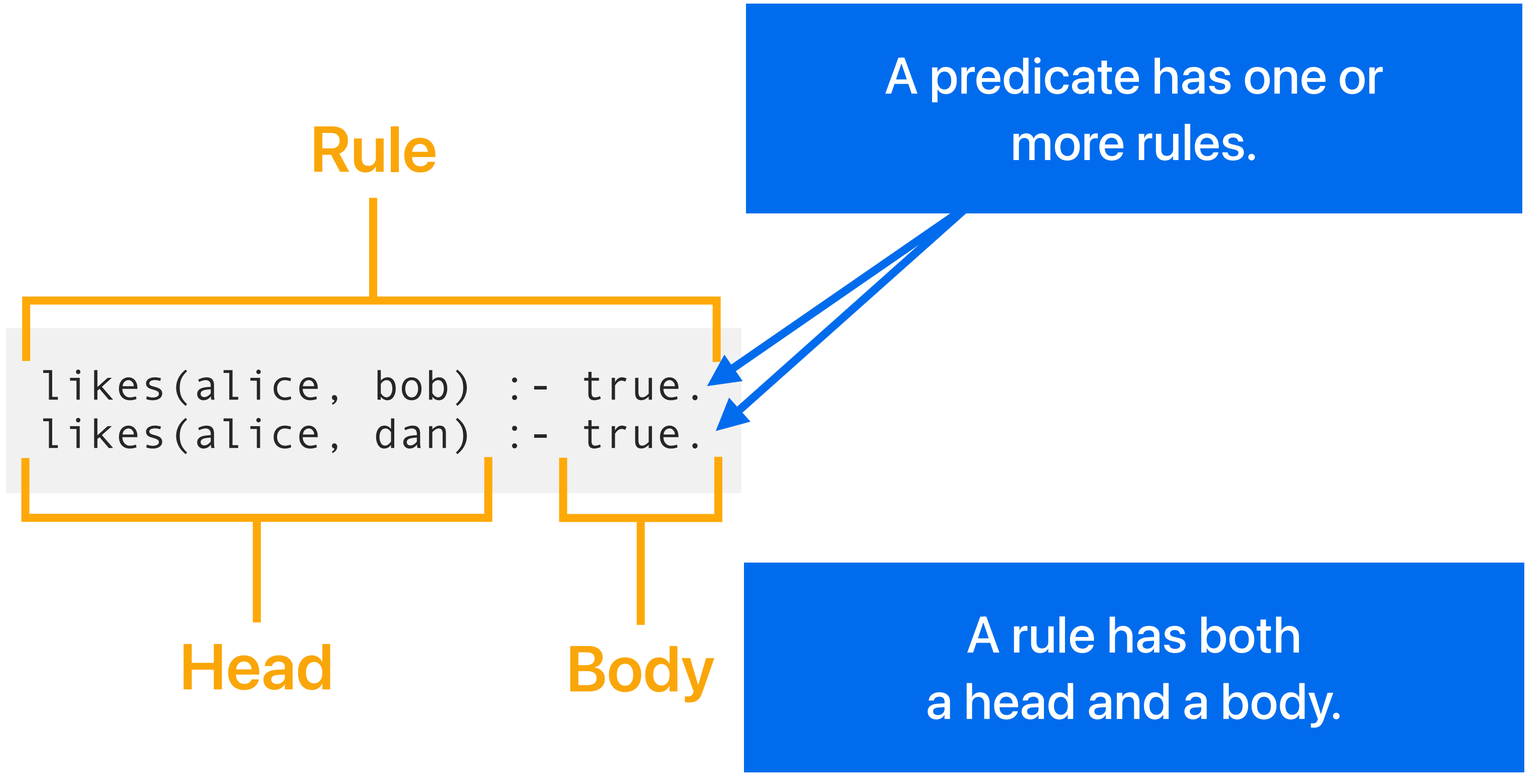
Here, “likes” is a predicate
with 2 arguments.

Anatomy of a Predicate

Prolog Programming 101

NEW WAY

Predicates



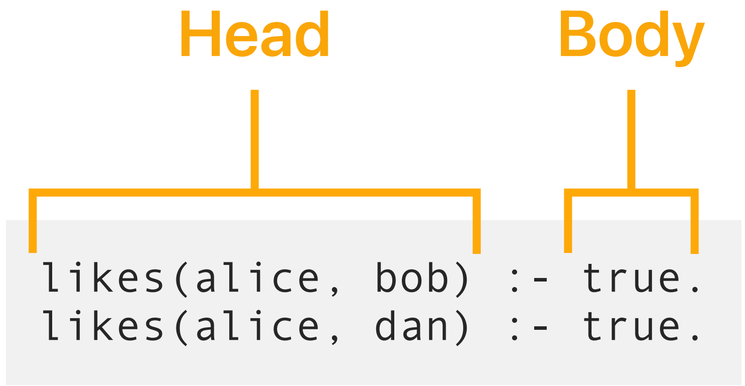
?- THE CLAUSE.

Anatomy of a Predicate

Prolog Programming 101

NEW WAY

Predicates



THE POINT:

A rule specifies:
"If my body is true,
then my head is true".

Anatomy of a Predicate

Prolog Programming 101

We're defining a "likes" predicate that documents who likes who!

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
```

First argument is the liker, second argument is the likee – according to our own interpretation!

Beyond all that, the **semantic meaning** of the code we write is *up to us!*

Prolog Programming 101

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
```

RECAP:

A predicate:

- has a ***name*** and zero or more ***arguments***.
- has one or more ***rules***.

And a rule has a ***head*** and ***body***.

Prolog Programming 102

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
```

Before we move on,
let's add more data!

Prolog Programming 1

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
```

Predicates are data.
They're how you *inform*
Prolog what is true.

You don't tell Prolog *how* to
find the truth.
You tell it *what* the truth *is*.

What *is* the truth?
Anything you want –
by your own definition.

Here we are defining a
"like" relationship between
many "persons".

Prolog doesn't "know" that
the first slot is the liker,
or that the second is the liked.
It just sees raw data!

Prolog Programming 102

Let's get more interesting.

Predicates also don't have to be "hardcoded" data.

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
```

Let's write a rule for Dan.
Dan only likes people who Alice likes.

Prolog Programming 102

Wow!
A body that isn't always true!

Dan only likes people who Alice likes.
This rule *depends* on other rules!

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.

likes(dan, P) :-
  likes(alice, P).
```

Let's move on to the next section
to see the juicy implications.

Variables start with
a capital letter
or an underscore

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

To start querying,
let's put our example code
in a new file...

Query Prompt

```
$ swipl
?- [my-file].
true.
?-
```

...and load it into
a Prolog implementation!
(Here we use SWI-Prolog)

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
$ swipl
?- [my-file].
true.
?-
```

Once at the query prompt,
we can start doing
what Prologgers **love** doing:
Asking questions.

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?-
```

First question:
Who does Alice like?

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
```

First question:
Who does Alice like?

First we type our query...

Variables start with
a capital letter
or with an underscore.

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob
```

First question:
Who does Alice like?

First we type our query...

...and then we hit enter.
Prolog found an answer for us!

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob ;
X = dan.
?-
```

First question:
Who does Alice like?

If you hit semicolon (or n), Prolog will keep searching for answers.

Prolog reports:
Alice also likes dan!

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob ;
X = dan.
?-
```

First question:
Who does Alice like?

At this point there is nothing left to search.

This is why you see the ?- prompt again. Prolog is ready for more!

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob ;
X = dan.
?- likes(P, dan).
```

Second question:
Who likes Dan?

Variables can appear in
any position!! 🤯
It's a predicate, not a function 😎

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob ;
X = dan.
?- likes(P, dan).
P = alice ;
P = carly.
?-
```

Second question:
Who likes Dan?

Alice and Carly like Dan –
just as we specified
in the code.

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob ;
X = dan.
?- likes(P, dan).
P = alice ;
P = carly.
?- likes(dan, Y).
```

Last question:
Who does Dan like?

Prolog Programming 103

my-file.pl

```
likes(alice, bob) :- true.
likes(alice, dan) :- true.
likes(bob, alice) :- true.
likes(bob, carly) :- true.
likes(carly, dan) :- true.
likes(dan, P) :-
  likes(alice, P).
```

Query Prompt

```
?- likes(alice, X).
X = bob ;
X = dan.
?- likes(P, dan).
P = alice ;
P = carly.
?- likes(dan, Y).
Y = bob ;
Y = dan
```

Last question:
Who does Dan like?

Alice likes Bob,
so Dan does too.

Alice likes Dan,
so Dan likes himself too!

There’s so much more power Prolog has to offer! But we’ll end here for now.

Conclusion

1. Prolog is useful!
2. Still alive and kicking after 50 years
3. A small taste of the power of prolog

Further Reading

- Learn Prolog in Y minutes
  https://learnxinyminutes.com/docs/prolog/

- Write a User Permissions System in 5 lines of Prolog
  https://dev.to/theclause/write-a-user-permissions-system-in-5-lines-of-prolog-mof

- The Power of Prolog
  https://www.metalevel.at/prolog