Announcements

*Project 4 due Thursday 11/21 @ 11:59pm.*
Announcements

• Project 4 due Thursday 11/21 @ 11:59pm.
• Extra reader office hours in 405 Soda this week.
Announcements

• Project 4 due Thursday 11/21 @ 11:59pm.

• Extra reader office hours in 405 Soda this week.
  • Wednesday: 5:30pm–7pm
  • Thursday: 5:30pm–7pm
Announcements

• Project 4 due Thursday 11/21 @ 11:59pm.
• Extra reader office hours in 405 Soda this week.
  • Wednesday: 5:30pm–7pm
  • Thursday: 5:30pm–7pm
• Homework 10 due Tuesday 11/26 @ 11:59pm.
Announcements

• Project 4 due Thursday 11/21 @ 11:59pm.
• Extra reader office hours in 405 Soda this week.
  • Wednesday: 5:30pm–7pm
  • Thursday: 5:30pm–7pm
• Homework 10 due Tuesday 11/26 @ 11:59pm.
• Recursive art contest entries will be due Monday 12/2 @ 11:59pm (After Thanksgiving).
Declarative Languages
Databases

http://www.headfirstlabs.com/sql_hands_on/
Databases

A table is a collection of records, which are tuples of values organized in columns.

http://www.headfirstlabs.com/sql_hands_on/
Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have methods for adding, editing, and retrieving records.
Databases

A table is a collection of records, which are tuples of values organized in columns. Databases store tables and have methods for adding, editing, and retrieving records. The Structured Query Language (SQL) is perhaps the most widely used programming language.

http://www.headfirstlabs.com/sql_hands_on/
Databases

A table is a collection of records, which are tuples of values organized in columns. Databases store tables and have methods for adding, editing, and retrieving records. The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

http://www.headfirstlabs.com/sql_hands_on/
Databases

A table is a collection of records, which are tuples of values organized in columns. Databases store tables and have methods for adding, editing, and retrieving records. The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

SQL is an example of a declarative programming language. It separates what to compute from how it is computed. The language interpreter is free to compute the result in any way it deems appropriate.

```
<table>
<thead>
<tr>
<th>toy_id</th>
<th>toy</th>
<th>color</th>
<th>cost</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>whiffleball</td>
<td>yellow</td>
<td>2.20</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>frisbee</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
<tr>
<td>10</td>
<td>yoyo</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
</tbody>
</table>
```
Databases

A table is a collection of records, which are tuples of values organized in columns. Databases store tables and have methods for adding, editing, and retrieving records. The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

Databases store tables and have methods for adding, editing, and retrieving records. The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

Each row is a record
Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

Databases store tables and have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

SQL is an example of a declarative programming language.

![Table example](http://www.headfirstlabs.com/sql_hands_on/)

<table>
<thead>
<tr>
<th>toy_id</th>
<th>toy</th>
<th>color</th>
<th>cost</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>whiffleball</td>
<td>yellow</td>
<td>2.20</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>frisbee</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
<tr>
<td>10</td>
<td>yoyo</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Each row is a record
Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

SQL is an example of a declarative programming language.

It separates what to compute from how it is computed.

Each row is a record

<table>
<thead>
<tr>
<th>toy_id</th>
<th>toy</th>
<th>color</th>
<th>cost</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>whiffleball</td>
<td>yellow</td>
<td>2.20</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>frisbee</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
<tr>
<td>10</td>
<td>yoyo</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
</tbody>
</table>

http://www.headfirstlabs.com/sql_hands_on/
Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

<table>
<thead>
<tr>
<th>toy_id</th>
<th>toy</th>
<th>color</th>
<th>cost</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>whiffleball</td>
<td>yellow</td>
<td>2.20</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>frisbee</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
<tr>
<td>10</td>
<td>yoyo</td>
<td>yellow</td>
<td>1.50</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Each row is a record

SQL is an example of a declarative programming language.

It separates what to compute from how it is computed.

The language interpreter is free to compute the result in any way it wants.
Declarative Programming
Declarative Programming

Characteristics of declarative languages:
Declarative Programming

Characteristics of declarative languages:

- A "program" is a description of the desired solution.
Declarative Programming

Characteristics of declarative languages:

• A "program" is a description of the desired solution.
• The interpreter figures out how to generate such a solution.
Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:
Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
Declarative Programming

Characteristics of **declarative languages**:
- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:
- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.
Declarative Programming

Characteristics of declarative languages:
• A "program" is a description of the desired solution.
• The interpreter figures out how to generate such a solution.

In imperative languages such as Python & Scheme:
• A "program" is a description of computational processes.
• The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.
Declarative Programming

Characteristics of **declarative languages**:  
- A "program" is a description of the desired solution.  
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:  
- A "program" is a description of computational processes.  
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.  
Declarative languages often handle only some subset of problems.
Declarative Programming

Characteristics of **declarative languages**:  
• A "program" is a description of the desired solution.  
• The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:  
• A "program" is a description of computational processes.  
• The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.  
Declarative languages often handle only some subset of problems.
Declarative Programming

Characteristics of declarative languages:
• A "program" is a description of the desired solution.
• The interpreter figures out how to generate such a solution.

In imperative languages such as Python & Scheme:
• A "program" is a description of computational processes.
• The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.
Declarative languages often handle only some subset of problems.

Solve cool problems

as long as they are small
Declarative Programming

Characteristics of declarative languages:
• A "program" is a description of the desired solution.
• The interpreter figures out how to generate such a solution.

In imperative languages such as Python & Scheme:
• A "program" is a description of computational processes.
• The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Solve cool problems
as long as they are small

Limited problem solving
on large-scale datasets
Declarative Programming

Characteristics of **declarative languages:**
- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:
- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

- **Solve cool problems** as long as they are small
- **Limited problem solving** on large-scale datasets
Declarative Programming

Characteristics of declarative languages:
• A "program" is a description of the desired solution.
• The interpreter figures out how to generate such a solution.

In imperative languages such as Python & Scheme:
• A "program" is a description of computational processes.
• The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Solve cool problems as long as they are small.

Limited problem solving on large-scale datasets.

Most applications
The Logic Language
The Logic Language
The Logic Language

The *Logic* language is invented for this course.
The Logic Language

The Logic language is invented for this course.
* Based on the Scheme project with ideas from Prolog (1972).
The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
The Logic Language

The *Logic* language is invented for this course.

* Based on the Scheme project with ideas from Prolog (1972).
* Expressions are facts or queries, which contain relations.
* Expressions and relations are Scheme lists.
The Logic Language

The Logic language is invented for this course.
• Based on the Scheme project with ideas from Prolog (1972).
• Expressions are facts or queries, which contain relations.
• Expressions and relations are Scheme lists.
• For example, (likes john dogs) is a relation.
The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.
- For example, *(likes john dogs)* is a relation.
- Implementation fits on a single sheet of paper (next lecture).
The Logic Language

The Logic language is invented for this course.
• Based on the Scheme project with ideas from Prolog (1972).
• Expressions are facts or queries, which contain relations.
• Expressions and relations are Scheme lists.
• For example, (likes john dogs) is a relation.
• Implementation fits on a single sheet of paper (next lecture).

Today's theme:
The Logic Language

The Logic language is invented for this course.
• Based on the Scheme project with ideas from Prolog (1972).
• Expressions are facts or queries, which contain relations.
• Expressions and relations are Scheme lists.
• For example, (likes john dogs) is a relation.
• Implementation fits on a single sheet of paper (next lecture).

Today's theme:
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

• A relation is a Scheme list.
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

\[
\text{logic}\text{> (fact (parent delano herbert))}
\]
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
```

Diagram:
```
Abraham -- Delano
          /   \     
         /     \    
  Barack   Clinton Herbert
```
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

```scheme
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
```
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:
• A relation is a Scheme list.
• A fact expression is a Scheme list of relations.

logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
Relations are Not Procedure Calls
Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.
Relations are Not Procedure Calls

In Logic, a relation is **not** a call expression.

- *Scheme*: the expression \((\text{abs} -3)\) calls \(\text{abs}\) on \(-3\). It returns 3.
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.

• Scheme: the expression \((\text{abs } -3)\) calls \(\text{abs}\) on \(-3\). It returns 3.
• Logic: \((\text{abs } -3 3)\) asserts that \(\text{abs}\) of \(-3\) is 3.
Relations are Not Procedure Calls

In Logic, a relation is **not** a call expression.

*Scheme:* the expression \((\text{abs} -3)\) calls \(\text{abs}\) on \(-3\). It returns 3.

*Logic:* \((\text{abs} -3 3)\) asserts that \(\text{abs}\) of \(-3\) is 3.

To assert that \(1 + 2 = 3\), we use a relation: \((\text{add} 1 2 3)\)
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.

• Scheme: the expression \((\text{abs } -3)\) calls \(\text{abs}\) on \(-3\). It returns 3.

• Logic: \((\text{abs } -3 3)\) asserts that \(\text{abs}\) of \(-3\) is 3.

To assert that \(1 + 2 = 3\), we use a relation: \((\text{add } 1 2 3)\)

We can ask the Logic interpreter to complete relations based on known facts.
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.

• Scheme: the expression (abs -3) calls abs on -3. It returns 3.
• Logic: (abs -3 3) asserts that abs of -3 is 3.

To assert that $1 + 2 = 3$, we use a relation: (add 1 2 3)

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3)
Relations are Not Procedure Calls

In Logic, a relation is **not** a call expression.

- *Scheme*: the expression `(abs -3)` calls `abs` on `-3`. It returns `3`.
- *Logic*: `(abs -3 3)` asserts that `abs` of `-3` is `3`.

To assert that `1 + 2 = 3`, we use a relation: `(add 1 2 3)`

We can ask the Logic interpreter to complete relations based on known facts.

```
(add ? 2 3) 1
```
Relations are Not Procedure Calls

In Logic, a relation is **not** a call expression.

- *Scheme*: the expression `(abs -3)` calls `abs` on -3. It returns 3.
- *Logic*: `(abs -3 3)` asserts that `abs` of -3 is 3.

To assert that $1 + 2 = 3$, we use a relation: `(add 1 2 3)`

We can ask the Logic interpreter to complete relations based on known facts.

```
(add ? 2 3)  1
(add 1 ? 3)
```
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.

• Scheme: the expression (abs -3) calls abs on -3. It returns 3.

• Logic: (abs -3 3) asserts that abs of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: (add 1 2 3)

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3) 1

(add 1 ? 3) 2
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.

• Scheme: the expression (abs -3) calls abs on -3. It returns 3.
• Logic: (abs -3 3) asserts that abs of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: (add 1 2 3)

We can ask the Logic interpreter to complete relations based on known facts.

```
(add ? 2 3)   1
(add 1 ? 3)   2
(add 1 2 ?)
```
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.
- *Scheme*: the expression `(abs -3)` calls `abs` on `-3`. It returns 3.
- *Logic*: `(abs -3 3)` asserts that `abs` of `-3` is `3`.

To assert that `1 + 2 = 3`, we use a relation: `(add 1 2 3)`

We can ask the Logic interpreter to complete relations based on known facts.

```
(add ? 2 3) 1
(add 1 ? 3) 2
(add 1 2 ?) 3
```
Relations are Not Procedure Calls

In Logic, a relation is **not** a call expression.

- **Scheme**: the expression *(abs -3)* calls *abs* on -3. It returns 3.
- **Logic**: *(abs -3 3)* asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: *(add 1 2 3)*

We can ask the Logic interpreter to complete relations based on known facts.

*(add ? 2 3)*

*(add 1 ? 3)*

*(add 1 2 ?)*

*( ? 1 2 3)*
Relations are Not Procedure Calls

In Logic, a relation is not a call expression.

- **Scheme**: the expression `(abs -3)` calls `abs` on -3. It returns 3.
- **Logic**: `(abs -3 3)` asserts that `abs` of -3 is 3.

To assert that `1 + 2 = 3`, we use a relation: `(add 1 2 3)`

We can ask the Logic interpreter to complete relations based on known facts.

```
(add ? 2 3) 1
(add 1 ? 3) 2
(add 1 2 ?) 3
(_, 1 2 3) add
```
Queries
A query contains one or more relations that may contain variables.
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with ?
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with \(?\)

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
```
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with \(?\)

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
```
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with ?

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
```
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with ?

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
```

A variable can have any name

```
Eisenhower

Fillmore

Abraham Delano Grover

Barack Clinton Herbert
```
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with ?

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
```

A variable can have any name
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with `?`

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
Success!
```

A variable can have any name.
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with ?

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
Success!
puppy: barack
```

A variable can have any name
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with `?`.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name.
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with ?

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name

Each line is an assignment of variables to values
Queries

A query contains one or more relations that may contain variables.

Variables are symbols starting with ?

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name

Each line is an assignment of variables to values

(Demo)
Compound Facts and Queries
Compound Facts

- Eisenhower
  - Fillmore
    - Abraham
    - Delano
    - Grover
    - Barack
    - Clinton
    - Herbert
Compound Facts

A fact can include multiple relations and variables as well.
Compound Facts

A fact can include multiple relations and variables as well.

\[(\text{fact} \ <\text{conclusion}> \ <\text{hypothesis}_0> \ <\text{hypothesis}_1> \ \ldots \ <\text{hypothesis}_N>)\]
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis_0> <hypothesis_1> ... <hypothesis_N>)

Means <conclusion> is true if all the <hypothesis_K> are true.
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis₀> <hypothesis₁> ... <hypothesisₙ>)

Means <conclusion> is true if all the <hypothesisₖ> are true.

Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis₀> <hypothesis₁> ... <hypothesisₙ>)

Means <conclusion> is true if all the <hypothesisₖ> are true.


logic> (query (child herbert delano))
Compound Facts

A fact can include multiple relations and variables as well.

\[(\text{fact } \langle\text{conclusion}\rangle \ \langle\text{hypothesis}_0\rangle \ \langle\text{hypothesis}_1\rangle \ \ldots \ \langle\text{hypothesis}_N\rangle)\]

Means \(\langle\text{conclusion}\rangle\) is true if all the \(\langle\text{hypothesis}_K\rangle\) are true.

\[\text{logic} > (\text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c))\]

\[\text{logic} > (\text{query} \ (\text{child} \ \text{herbert} \ \text{delano}))\]

Success!

\[\text{Eisenhower} \quad \downarrow \quad \text{Fillmore} \quad \downarrow \quad \text{Abraham} \quad \downarrow \quad \text{Delano} \quad \downarrow \quad \text{Grover} \]

\[\text{Barack} \quad \downarrow \quad \text{Clinton} \quad \downarrow \quad \text{Herbert} \]
Compound Facts

A fact can include multiple relations and variables as well.

(\text{fact } \text{<conclusion>} \text{<hypothesis}_0\text{<hypothesis}_1\ldots \text{<hypothesis}_N\text{)})

Means \text{<conclusion>} is true if all the \text{<hypothesis}_k\text{ are true.}

\text{logic} \ (\text{fact (child } ?c \ ?p) \ (parent \ ?p \ ?c))

\text{logic} \ (\text{query (child herbert delano)})
\text{Success!}

\text{logic} \ (\text{query (child eisenhower clinton)})
Compound Facts

A fact can include multiple relations and variables as well.

\[(\text{fact} \ <\text{conclusion}> \ <\text{hypothesis}_0> \ <\text{hypothesis}_1> \ ... \ <\text{hypothesis}_N>)\]

Means \(<\text{conclusion}>\) is true if all the \(<\text{hypothesis}_K>\) are true.

\[
\begin{align*}
\text{logic} & > (\text{fact} (\text{child} ?c ?p) (\text{parent} ?p ?c)) \\
\text{logic} & > (\text{query} (\text{child} \text{ herbert delano})) \\
\text{Success!} &  \\
\text{logic} & > (\text{query} (\text{child} \text{ eisenhower clinton})) \\
\text{Failure.} &
\end{align*}
\]
A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis₀> <hypothesis₁> ... <hypothesisₙ>)

Means <conclusion> is true if all the <hypothesisₖ> are true.


logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.

logic> (query (child ?kid fillmore))
A fact can include multiple relations and variables as well.

\( \text{fact} \ <\text{conclusion}> \ <\text{hypothesis}_0> \ <\text{hypothesis}_1> \ldots \ <\text{hypothesis}_N> \)

Means \( <\text{conclusion}> \) is true if all the \( <\text{hypothesis}_K> \) are true.

\[ \text{logic} > (\text{fact} (\text{child} \ ?c \ ?p) (\text{parent} \ ?p \ ?c)) \]
\[ \text{logic} > (\text{query} (\text{child} \ \text{herbert} \ \text{delano})) \]
\text{Success!} \]

\[ \text{logic} > (\text{query} (\text{child} \ \text{eisenhower} \ \text{clinton})) \]
\text{Failure.} \]

\[ \text{logic} > (\text{query} (\text{child} \ ?\text{kid} \ \text{fillmore})) \]
\text{Success!}
Compound Facts

A fact can include multiple relations and variables as well.

\[(\text{fact } \text{<conclusion>} \text{<hypothesis\textsubscript{0}> } \text{<hypothesis\textsubscript{1}> } \ldots \text{<hypothesis\textsubscript{N}>})\]

Means \text{<conclusion>} is true if all the \text{<hypothesis\textsubscript{K}>} are true.

\begin{verbatim}
logic> (query (child herbert delano))
Success!
logic> (query (child eisenhower clinton))
Failure.
logic> (query (child ?kid fillmore))
Success!
kid: abraham
\end{verbatim}
Compound Facts

A fact can include multiple relations and variables as well.

\[(\text{fact } \text{<conclusion>} \ \text{<hypothesis}_0> \ \text{<hypothesis}_1> \ \ldots \ \text{<hypothesis}_N>\)\]

Means \text{<conclusion>} is true if all the \text{<hypothesis}_K> are true.

\[
\text{logic} > (\text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c))
\]

\[
\text{logic} > (\text{query} \ (\text{child} \ \text{herbert} \ \text{delano}))
\text{Success!}
\]

\[
\text{logic} > (\text{query} \ (\text{child} \ \text{eisenhower} \ \text{clinton}))
\text{Failure}.
\]

\[
\text{logic} > (\text{query} \ (\text{child} \ ?\text{kid} \ \text{fillmore}))
\text{Success!}
\]

\text{kid: abraham}
\text{kid: delano}

\[
\text{logic} > (\text{query} \ (\text{child} \ ?\text{kid} \ \text{fillmore}))
\text{Success!}
\]

\text{kid: abraham}
\text{kid: delano}
Compound Facts

A fact can include multiple relations and variables as well.

\[(\text{fact} \ <\text{conclusion}> \ <\text{hypothesis}_0> \ <\text{hypothesis}_1> \ ... \ <\text{hypothesis}_N>)\]

Means \(<\text{conclusion}>\) is true if all the \(<\text{hypothesis}_K>\) are true.

\begin{align*}
\text{logic}> (\text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c))\\
\text{logic}> (\text{query} \ (\text{child} \ \text{herbert} \ \text{delano}))\\
\text{Success!}\\
\text{logic}> (\text{query} \ (\text{child} \ \text{eisenhower} \ \text{clinton}))\\
\text{Failure.}\\
\text{logic}> (\text{query} \ (\text{child} \ ?\text{kid} \ \text{fillmore}))\\
\text{Success!}\\
\text{kid}: \ \text{abraham}\\
\text{kid}: \ \text{delano}\\
\text{kid}: \ \text{grover}
\end{align*}
Compound Queries

Eisenhower

Fillmore

Abraham

Delano

Grover

Barack

Clinton

Herbert
Compound Queries

An assignment must satisfy all relations in a query.
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query } \langle \text{relation}_0 \rangle \ \langle \text{relation}_1 \rangle \ \ldots \ \langle \text{relation}_N \rangle)\]
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation_0> <relation_1> ... <relation_N>)

is satisfied if all the <relation_K> are true.
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query } \text{<relation}_0\text{> } \text{<relation}_1\text{> } \ldots \text{<relation}_N\text{>})\]

is satisfied if all the \text{<relation}_K\text{>} are true.

\[
\text{logic} > (\text{fact} (\text{child} ?c ?p) (\text{parent} ?p ?c))
\]

Diagram:
- Top: Eisenhower
- Middle left: Fillmore
  - Left: Abraham
  - Right: Delano
- Middle right: Grover
  - Right: Herbert
  - Left: Clinton
  - Left: Barack
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query} \ <\text{relation}_0> \ <\text{relation}_1> \ldots \ <\text{relation}_N>)\]

is satisfied if all the \(<\text{relation}_k>\) are true.

\[
\text{logic} > (\text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c))
\]

\[
\text{logic} > (\text{query} \ (\text{parent} \ ?\text{grampa} \ ?\text{kid})
\]
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query } \langle \text{relation}_0 \rangle \ < \langle \text{relation}_1 \rangle \ ... \ < \langle \text{relation}_n \rangle)\]

is satisfied if all the \(\langle \text{relation}_k \rangle\) are true.

\[\text{logic} \; \langle \text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c) \rangle\]

\[\text{logic} \; \langle \text{query} \ (\text{parent} \ ?\text{grampa} \ ?\text{kid}) \ (\text{child} \ \text{clinton} \ ?\text{kid}) \rangle\]
Compound Queries

An assignment must satisfy all relations in a query.

\[(query \ <relation_0> \ <relation_1> \ ... \ <relation_N>)\]

is satisfied if all the \(<relation_K>\) are true.

\[
\text{logic} > (\text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c))
\]

\[
\text{logic} > (\text{query} \ (\text{parent} \ ?grampa \ ?kid) \ (\text{child} \ clinton \ ?kid))
\]

Success!
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query} \ <\text{relation}_0>\ <\text{relation}_1>\ ...\ <\text{relation}_n>)\]

is satisfied if all the \(<\text{relation}_k>\) are true.


logic> (query (parent ?grampa ?kid) (child clinton ?kid))

Success!

grampa: fillmore kid: abraham
Compound Queries

An assignment must satisfy all relations in a query.

\[(query \ <relation_0> \ <relation_1> \ ... \ <relation_N>)\]

is satisfied if all the \( <relation_K> \) are true.

\[
\text{logic} \ (\text{fact} \ (\text{child} \ ?c \ ?p) \ (\text{parent} \ ?p \ ?c))
\]

\[
\text{logic} \ (\text{query} \ (\text{parent} \ ?\text{grampa} \ ?\text{kid})
\quad (\text{child} \ \text{clinton} \ ?\text{kid}))
\]

Success!

grampa: fillmore  kid: abraham

\[
\text{logic} \ (\text{query} \ (\text{child} \ ?y \ ?x))
\]
**Compound Queries**

An assignment must satisfy all relations in a query.

\[
\text{(query } \text{relation}_0 \text{ relation}_1 \ldots \text{ relation}_N\text{)}
\]

is satisfied if all the \text{relation}_K are true.

\[
\text{logic}\ (\text{fact}\ (\text{child} \ ?c \ ?p)\ (\text{parent} \ ?p \ ?c))
\]

\[
\text{logic}\ (\text{query}\ (\text{parent} \ ?\text{grampa} \ ?\text{kid})
\qquad (\text{child}\ \text{clinton} \ ?\text{kid}))
\]

Success!

grampa: fillmore    kid: abraham

\[
\text{logic}\ (\text{query}\ (\text{child} \ ?y \ ?x)
\qquad (\text{child} \ ?x \ \text{eisenhower}))
\]
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query } \langle \text{relation}_0 \rangle \ \langle \text{relation}_1 \rangle \ldots \langle \text{relation}_N \rangle)\]

is satisfied if all the \(\langle \text{relation}_K \rangle\) are true.

\[\text{logic}\!>\! \text{(fact (child ?c ?p) (parent ?p ?c))}\]

\[\text{logic}\!>\! \text{(query (parent ?grampa ?kid) (child clinton ?kid))}\]

Success!

grampa: fillmore    kid: abraham

\[\text{logic}\!>\! \text{(query (child ?y ?x) (child ?x eisenhower))}\]

Success!

\begin{center}
\begin{tikzpicture}
  \node (eisenhower) at (4,0) {Eisenhower};
  \node (fillmore) at (3,-1) {Fillmore};
  \node (abraham) at (2,-2) {Abraham};
  \node (clinton) at (4,-2) {Clinton};
  \node (delano) at (5,-2) {Delano};
  \node (grover) at (6,-2) {Grover};
  \node (barack) at (2,-3) {Barack};
  \node (herbert) at (4,-3) {Herbert};

  \draw (eisenhower) -- (fillmore);
  \draw (fillmore) -- (abraham);
  \draw (fillmore) -- (clinton);
  \draw (fillmore) -- (delano);
  \draw (fillmore) -- (grover);
  \draw (abraham) -- (barack);
  \draw (clinton) -- (herbert);
\end{tikzpicture}
\end{center}
Compound Queries

An assignment must satisfy all relations in a query.

\[(\text{query } \text<relation}_0\text> \text<relation}_1\text> \ldots \text<relation}_N\text>\)

is satisfied if all the \text<relation>_K\text> are true.

\begin{verbatim}
logic> (query (parent ?grampa ?kid) (child clinton ?kid))
Success!
grampa: fillmore kid: abraham
logic> (query (child ?y ?x) (child ?x eisenhower))
Success!
y: abraham x: fillmore
\end{verbatim}
Compound Queries

An assignment must satisfy all relations in a query.

$$(\text{query } <\text{relation}_0> <\text{relation}_1> \ldots <\text{relation}_N>)$$

is satisfied if all the $<\text{relation}_K>$ are true.

\[
\text{logic} > (\text{fact} (\text{child} ?c ?p) (\text{parent} ?p ?c))
\]

\[
\text{logic} > (\text{query} (\text{parent} ?\text{grampa} ?\text{kid})
\quad (\text{child} \text{clinton} ?\text{kid}))
\]

**Success!**
grampa: fillmore  kid: abraham

\[
\text{logic} > (\text{query} (\text{child} ?y ?x)
\quad (\text{child} ?x \text{eisenhower}))
\]

**Success!**
y: abraham  x: fillmore
y: delano  x: fillmore
An assignment must satisfy all relations in a query.

\[
(\text{query } <\text{relation}_0> <\text{relation}_1> \ldots <\text{relation}_N>)
\]

is satisfied if all the \(<\text{relation}_K>\) are true.

```
```

```
logic> (query (parent ?grampa ?kid)
            (child clinton ?kid))
Success!
grampa: fillmore  kid: abraham
```

```
logic> (query (child ?y ?x)
            (child ?x eisenhower))
Success!
y: abraham   x: fillmore
y: delano    x: fillmore
y: grover    x: fillmore
```
Recursive Facts
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

logic> (fact (ancestor ?a ?y) (parent ?a ?y))
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

logic> (fact (ancestor ?a ?y) (parent ?a ?y))
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

logic> (fact (ancestor ?a ?y) (parent ?a ?y))

logic> (query (ancestor ?a herbert))
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

logic> (fact (ancestor ?a ?y) (parent ?a ?y))

logic> (query (ancestor ?a herbert))
Success!

![Family Tree Diagram]
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
```
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
```
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

logic> (fact (ancestor ?a ?y) (parent ?a ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack))
```
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))

logic> (query (ancestor ?a herbert))
Success!
```

```
| a: delano  |
| a: fillmore|
| a: eisenhower |
```

```
logic> (query (ancestor ?a barack) (ancestor ?a herbert))
```

```
Eisenhower
  
Fillmore
  
Abraham
  
Barack
  
Clinton
  
Herbert
```

```
Delano
  
Grover
```

16
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

```
logic> (query (ancestor ?a barack) (ancestor ?a herbert))
Success!
```

![Diagram showing a family tree with relationships between different names.]
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

```
logic> (query (ancestor ?a barack) (ancestor ?a herbert))
Success!
a: fillmore
```

![Diagram showing the recursive relationship between ancestors and descendants.](Diagram)
Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

\[
\text{logic}\> (\text{fact} \ (\text{ancestor} \ ?a \ ?y) \ (\text{parent} \ ?a \ ?y))
\text{logic}\> (\text{fact} \ (\text{ancestor} \ ?a \ ?y) \ (\text{parent} \ ?a \ ?z) \ (\text{ancestor} \ ?z \ ?y))
\]

\[
\text{logic}\> (\text{query} \ (\text{ancestor} \ ?a \ \text{herbert}))
\text{Success!}
a: \ delano
a: \ fillmore
a: \ eisenhower
\]

\[
\text{logic}\> (\text{query} \ (\text{ancestor} \ ?a \ \text{barack}) \ (\text{ancestor} \ ?a \ \text{herbert}))
\text{Success!}
a: \ fillmore
a: \ eisenhower
\]

![Family Tree Diagram]
Searching to Satisfy Queries
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```logic
logic> (query (ancestor ?a herbert))
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

logic> (query (ancestor ?a herbert))
Success!
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!

a: delano
a: fillmore
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!

a: delano
a: fillmore
a: eisenhower

logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!

a: delano
a: fillmore
a: eisenhower
```

```
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```

```
(pARENT delano herbert) ; (1), a simple fact
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))

(parent delano herbert) ; (1), a simple fact
(ancestor delano herbert) ; (2), from (1) and the 1st ancestor fact
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

```
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```

```
(parent delano herbert)   ; (1), a simple fact
(ancestor delano herbert) ; (2), from (1) and the 1st ancestor fact
(parent fillmore delano) ; (3), a simple fact
```
Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
```

```
a: delano
a: fillmore
a: eisenhower
```

```
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```

```
(parent delano herbert) ; (1), a simple fact
(ancestor delano herbert) ; (2), from (1) and the 1st ancestor fact
(parent fillmore delano) ; (3), a simple fact
(ancestor fillmore herbert) ; (4), from (2), (3), & the 2nd ancestor fact
```
Hierarchical Facts
Hierarchical Facts
Hierarchical Facts

Relations can contain relations in addition to symbols.
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
Hierarchical Facts

Relations can contain relations in addition to symbols.

```logic
(logic> (fact (dog (name abraham) (color white))))
(logic> (fact (dog (name barack) (color tan))))
```
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
Hierarchical Facts

Relations can contain relations in addition to symbols.

\[
\text{logic} > (\text{fact} (\text{dog} (\text{name abraham}) (\text{color white}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name barack}) (\text{color tan}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name clinton}) (\text{color white}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name delano}) (\text{color white})))
\]
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
```
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.
Hierarchical Facts

Relations can contain relations in addition to symbols.

\[
\text{logic} > (\text{fact} (\text{dog} (\text{name abraham}) (\text{color white}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name barack}) (\text{color tan}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name clinton}) (\text{color white}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name delano}) (\text{color white}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name eisenhower}) (\text{color tan}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name fillmore}) (\text{color gray}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name grover}) (\text{color tan}))) \\
\text{logic} > (\text{fact} (\text{dog} (\text{name herbert}) (\text{color gray})))
\]

Variables can refer to symbols or whole relations.

\[
\text{logic} > (\text{query} (\text{dog} (\text{name clinton}) (\text{color} \ ?\text{color})))
\]
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))

Variables can refer to symbols or whole relations.

logic> (query (dog (name clinton) (color ?color)))
Success!

A
B
C
D
E
F
G
H
Hierarchical Facts

Relations can contain relations in addition to symbols.

```logic
> (fact (dog (name abraham) (color white)))
> (fact (dog (name barack) (color tan)))
> (fact (dog (name clinton) (color white)))
> (fact (dog (name delano) (color white)))
> (fact (dog (name eisenhower) (color tan)))
> (fact (dog (name fillmore) (color gray)))
> (fact (dog (name grover) (color tan)))
> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```logic
> (query (dog (name clinton) (color ?color)))
Success!
color: white
```
Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white
logic> (query (dog (name clinton) ?stats))
```
Hierarchical Facts

Relations can contain relations in addition to symbols.

logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))

Variables can refer to symbols or whole relations.

logic> (query (dog (name clinton) (color ?color)))
Success!
color: white

logic> (query (dog (name clinton) ?stats))
Success!
Hierarchical Facts

Relations can contain relations in addition to symbols.

```logic
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```logic
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white

logic> (query (dog (name clinton) ?stats))
Success!
stats: (color white)
```
Combining Multiple Data Sources
Combining Multiple Data Sources
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

```
logic> (query (dog (name ?x) (color ?fur))
```

![Diagram showing relationships between dogs A, B, C, D, E, F, G, and H. Dog E is connected to dog F, F is connected to dogs A, B, C, and D, and D is connected to dog G. Dog H is not connected to any other dog.]


Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

logic> (query (dog (name ?x) (color ?fur))
(ancestor ?y ?x)
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

```logic
> (query (dog (name ?x) (color ?fur))
  (ancestor ?y ?x)
  (dog (name ?y) (color ?fur)))
```
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

\[
\text{logic} > \text{(query (dog (name ?x) (color ?fur))}
\text{(ancestor ?y ?x)}
\text{(dog (name ?y) (color ?fur)))}
\]

Success!
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

logic> (query (dog (name ?x) (color ?fur))
  (ancestor ?y ?x)
  (dog (name ?y) (color ?fur)))

Success!

x: barack    fur: tan    y: eisenhower
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

logic> (query (dog (name ?x) (color ?fur))
   (ancestor ?y ?x)
   (dog (name ?y) (color ?fur)))

Success!

x: barack    fur: tan    y: eisenhower
x: clinton  fur: white  y: abraham
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

`logic> (query (dog (name ?x) (color ?fur))
            (ancestor ?y ?x)
            (dog (name ?y) (color ?fur)))
Success!

x: barack    fur: tan     y: eisenhower
x: clinton   fur: white   y: abraham
x: grover    fur: tan     y: eisenhower
Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

logic> (query (dog (name ?x) (color ?fur))
  (ancestor ?y ?x)
  (dog (name ?y) (color ?fur)))

Success!

x: barack   fur: tan   y: eisenhower
x: clinton  fur: white y: abraham
x: grover   fur: tan   y: eisenhower
x: herbert  fur: gray  y: fillmore