Guide to First-Order Logic Translations
Hi everybody!
In last Friday's lecture, we talked about how to translate statements from English into first-order logic.
Translating into logic is a skill that takes some practice to get used to, but once you get the hang of it, it’s actually not too bad - and honestly it can be a lot of fun!
In many ways, learning how to translate into first-order logic is like learning how to program.
You’ve got this crazy set of symbols and terms with precise meanings...
∀x. (P(x) ∨ R(x) →
  ∃y. (S(y) ∧ Q(x, y)))

...and the goal is combine them together in a way that says something interesting.
The good news is that, like programming, there are a lot of common patterns that come up time and time again in first-order logic.
Once you’ve gotten the handle on these patterns and the methodology of how to do a translation, you’ll find that it’s a lot easier to approach logic translations.
Let’s illustrate this with an analogy.
private int sumOf(int[] elems) {
    int result = 0;
    for (int i = 0; i < elems.length; i++) {
        result += elems[i];
    }
    return result;
}
This is a method that takes in an array of integers and returns the sum of the elements in that array.
Let's focus on this for loop.

```java
private int sumOf(int[] elems) {
    int result = 0;
    for (int i = 0; i < elems.length; i++) {
        result += elems[i];
    }
    return result;
}
```
If you've been programming for a while, you can look at this loop and pretty quickly read it as "loop over the elements of an array" loop.
There's actually a lot going on in this loop, though.
There's a variable declaration here that makes a new variable that tracks an index...
private int sumOf(int[] elems) {
    int result = 0;
    for (int i = 0; i < elems.length; i++) {
        result += elems[i];
    }
    return result;
}
private int sumOf(int[] elems) {
    int result = 0;
    for (int i = 0; i < elems.length; i++) {
        result += elems[i];
    }
    return result;
}
...and a test to see whether we've read everything that relies specifically on using the `<` operator and not other operators like `==` or `<=`.
When you're first learning to program, code like this can seem really, really complicated, but when you've been programming for a while you don't think about it that much.
It's just "idiomatic" code – you know what it does by sight even if you don't think too hard about what it means.
In many ways, first-order logic formulas are the same way.
∀p. (Person(p) →
  ∃q. (Person(q) ∧ p ≠ q ∧ Loves(p, q)
 )
 )

Here's a first-order logic formula from lecture. It objectively has a lot of symbols strewn throughout it.
∀p. (Person(p) → 
    ∃q. (Person(q) ∧ p ≠ q ∧ 
    Loves(p, q))
)
∀p. (Person(p) → 
   ∃q. (Person(q) ∧ p ≠ q ∧ 
       Loves(p, q))
 )
 )

If you tried to build this formula completely from scratch, it would be really challenging. However, if you know the patterns and how to string them together, this is a very natural formula to write.
\( \forall p. (\text{Person}(p) \rightarrow \\ \exists q. (\text{Person}(q) \land p \neq q \land \\ \text{Loves}(p, q) ) \\ ) \\ ) \\ )

This guide is designed to teach you what these common patterns are, how to combine them together, and how to use them to translate complicated statements.
\( \forall p. (\text{Person}(p) \rightarrow \exists q. (\text{Person}(q) \land p \neq q \land \text{Loves}(p, q)) ) \)

Think of it as a crash course in first-order logic programming.
∀p. (Person(p) →
    ∃q. (Person(q) ∧ p ≠ q ∧ Loves(p, q))
  )
)
Most of the time, when you’re writing statements in first-order logic, you’ll be making a statement of the form “every X has property Y” or “some X has property Y.”
Statements of these (usually) fall into one of four fundamental types of statements.
These four classes of statements are called *Aristotelian Forms*, since they were first described by Aristotle in his work "Prior Analytics"... though you don’t need to know that unless you want to show off at cocktail parties. ^_^
On Wednesday, we saw how to translate these statements into first-order logic. Here’s what we came up with.

“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]
"All $P$s are $Q$s."
$\forall x. (P(x) \rightarrow Q(x))$

"No $P$s are $Q$s."
$\forall x. (P(x) \rightarrow \neg Q(x))$

"Some $P$s are $Q$s."
$\exists x. (P(x) \land Q(x))$

"Some $P$s aren't $Q$s."
$\exists x. (P(x) \land \neg Q(x))$

In lecture we spent time talking about why $\forall$ gets paired with $\rightarrow$ and why $\exists$ gets paired with $\land$. We already talked in lecture about why this is, so we're not going to review it here. After all, our goal is to see how to use these patterns, not how to derive them.
“All Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. \ (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. \ (P(x) \land \neg Q(x)) \]

However, you **absolutely** should memorize these patterns. They’re like the “loop over an array” for loop pattern in Java, C, or C++ – they come up frequently and you ultimately want to get to the point where you can easily read and write them as a unit.
"All Ps are Qs."
\( \forall x. (P(x) \rightarrow Q(x)) \)

"No Ps are Qs."
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

"Some Ps are Qs."
\( \exists x. (P(x) \land Q(x)) \)

"Some Ps aren't Qs."
\( \exists x. (P(x) \land \neg Q(x)) \)

Now, let's see how we can use these four statements as building blocks for constructing larger statements.
Imagine that we have these predicates available to us.

- **"All Ps are Qs."**
  \[ \forall x. (P(x) \rightarrow Q(x)) \]
- **"No Ps are Qs."**
  \[ \forall x. (P(x) \rightarrow \neg Q(x)) \]
- **"Some Ps are Qs."**
  \[ \exists x. (P(x) \land Q(x)) \]
- **"Some Ps aren't Qs."**
  \[ \exists x. (P(x) \land \neg Q(x)) \]

**Available Predicates:**

- Orange(x)
- Cat(x)
- Fluffy(x)
and that we want to translate this statement into first-order logic.

"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

Every orange cat is fluffy.

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)
Let's see how we can use these formulas to help out our translation.

“All Ps are Qs.” 
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.” 
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.” 
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.” 
\[ \exists x. (P(x) \land \neg Q(x)) \]

Every orange cat is fluffy.

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)

Let's see how we can use these formulas to help out our translation.
“All Ps are Qs.”
∀x. (P(x) → Q(x))

“All Ps are Qs.”
∀x. (P(x) → Q(x))

“No Ps are Qs.”
∀x. (P(x) → ¬Q(x))

“No Ps are Qs.”
∀x. (P(x) → ¬Q(x))

“Some Ps are Qs.”
∃x. (P(x) ∧ Q(x))

“Some Ps aren't Qs.”
∃x. (P(x) ∧ ¬Q(x))

Every orange cat is fluffy.

Available Predicates:
Orange(x)
Cat(x)
Fluffy(x)

First, what kind of statement is this?
It seems to look a lot like this one – we’re saying that all objects of one kind (orange cats) are also of another kind (fluffy).

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)

Every orange cat is fluffy.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“All Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

Every orange cat is fluffy.

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)
we can start adding in a bit of structure to our first-order logic formula.

“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (x \text{ is an orange cat } \rightarrow x \text{ is fluffy}) \]

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)

...we can start adding in a bit of structure to our first-order logic formula.
From here, our goal is to keep replacing the remaining English statements in the formula with something in first-order logic that says the same thing.

Available Predicates:
Orange(x)
Cat(x)
Fluffy(x)
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (x \text{ is an orange cat } \rightarrow x \text{ is fluffy}) \]

Available Predicates:

- Orange(x)
- Cat(x)
- Fluffy(x)

For example, this part of the formula is easy to translate...
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \forall x. (x \text{ is an orange cat} \rightarrow x \text{ is fluffy}) \)

Available Predicates:

Orange(x)
Cat(x)
Fluffy(x)

...because we have a predicate that directly expresses this idea!
“All Ps are Qs.”
\(\forall x. (P(x) \rightarrow Q(x))\)

“Some Ps are Qs.”
\(\exists x. (P(x) \land Q(x))\)

“No Ps are Qs.”
\(\forall x. (P(x) \rightarrow \neg Q(x))\)

“Some Ps aren't Qs.”
\(\exists x. (P(x) \land \neg Q(x))\)

\(\forall x. (x \text{ is an orange cat} \rightarrow Fluffy(x))\)

Available Predicates:

Orange(x)
Cat(x)
Fluffy(x)

So let's go and snap that predicate in there.
Progress!
“All $Ps$ are $Qs$.”
$\forall x. \ (P(x) \rightarrow Q(x))$

“Some $Ps$ are $Qs$.”
$\exists x. \ (P(x) \land Q(x))$

“No $Ps$ are $Qs$.”
$\forall x. \ (P(x) \rightarrow \neg Q(x))$

“Some $Ps$ aren't $Qs$.”
$\exists x. \ (P(x) \land \neg Q(x))$

$\forall x. \ (x \text{ is an orange cat } \rightarrow Fluffy(x))$

**Available Predicates:**
- Orange($x$)
- Cat($x$)
- Fluffy($x$)
Well, we have two independent predicates – Orange\( (x) \) and Cat\( (x) \) – that each express a part of the idea. How can we combine them together?

\[
\forall x. (x \text{ is an orange cat} \rightarrow \text{Fluffy}(x))
\]

Available Predicates:

- Orange\( (x) \)
- Cat\( (x) \)
- Fluffy\( (x) \)
Let's begin by seeing how not to do this.

<table>
<thead>
<tr>
<th>“All Ps are Qs.”</th>
<th>“Some Ps are Qs.”</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀x. ( \text{P}(x) \rightarrow \text{Q}(x) )</td>
<td>∃x. ( \text{P}(x) \land \text{Q}(x) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“No Ps are Qs.”</th>
<th>“Some Ps aren't Qs.”</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀x. ( \text{P}(x) \rightarrow \neg \text{Q}(x) )</td>
<td>∃x. ( \text{P}(x) \land \neg \text{Q}(x) )</td>
</tr>
</tbody>
</table>

∀x. \( \text{x is an orange cat} \rightarrow \text{Fluffy}(x) \)

**Available Predicates:**

- \(\text{Orange}(x)\)
- \(\text{Cat}(x)\)
- \(\text{Fluffy}(x)\)

Let's begin by seeing how not to do this.
I'm going to put up our trusty warning indicators to show that what we're about to do is a really bad idea.

- "All Ps are Qs."
  \[\forall x. \ (P(x) \rightarrow Q(x))\]
- "Some Ps are Qs."
  \[\exists x. \ (P(x) \land Q(x))\]
- "No Ps are Qs."
  \[\forall x. \ (P(x) \rightarrow \neg Q(x))\]
- "Some Ps aren't Qs."
  \[\exists x. \ (P(x) \land \neg Q(x))\]

\[\forall x. \ (x \text{ is an orange cat} \rightarrow \text{Fluffy}(x))\]
Here’s something common we see people do that doesn’t work,

“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“All Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

Available Predicates:
Orange(x)
Cat(x)
Fluffy(x)
This superficially looks like it works correctly – it seems like it’s saying that x is a cat that’s orange.

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)
“All Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow Q(x)) \]

“All P's are Q's.”

“The problem is that it's not syntactically valid — it's the sort of mistake that would be a “compiler error” in many languages.”

“Some Ps are Qs.”
\[ \exists x. \ (P(x) \land Q(x)) \]

“No P's are Q's.”
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

“No P's are Q's.”

“Some P's aren't Q's.”
\[ \exists x. \ (P(x) \land \neg Q(x)) \]

“Some P's aren't Q's.”

Available Predicates:
Orange(x)  
Cat(x)  
Fluffy(x)

The problem is that it's not syntactically valid — it's the sort of mistake that would be a “compiler error” in many languages.
“All Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

“The reason this doesn’t work is that Orange and Cat are predicates – they take in objects and produce either true or false.”

“Some Ps are Qs.”
\[ \exists x. \ (P(x) \land Q(x)) \]

“Some Ps aren’t Qs.”
\[ \exists x. \ (P(x) \land \neg Q(x)) \]

\[ \forall x. \ (\text{Orange}(\text{Cat}(x)) \rightarrow \text{Fluffy}(x)) \]

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)
This means that the statement $\text{Cat}(x)$ evaluates to either “true” or “false.” Intuitively, it takes in an object and returns a boolean.
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

The problem is that \textit{Orange} expects that it will take in an object and return a boolean – but it's not being provided an object as input!
“All Ps are Qs.”
\[ \forall x. \ (P(x) \to Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. \ (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. \ (P(x) \to \lnot Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. \ (P(x) \land \lnot Q(x)) \]

⚠️ ∀x. \ (Orange(Cat(x)) \to Fluffy(x))

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)

This is the first-order logic equivalent of a type error.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

⚠️ ⚠️
\[ \forall x. (Orange(Cat(x)) \rightarrow Fluffy(x)) \]

Available Predicates:

Orange(x)  
Cat(x)  
Fluffy(x)

So even though this might at first glance seem right, it’s not actually legal... so we’re going to have to find some other way of expressing this idea!
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \forall x. (x \text{ is an orange cat} \rightarrow \text{Fluffy}(x)) \)

Available Predicates:

Orange(x)  
Cat(x)  
Fluffy(x)
We’re trying to express the idea that \( x \) is an orange cat.

\[
\forall x. \ (\text{Orange}(x) \rightarrow \text{Fluffy}(x))
\]

Available Predicates:

- \( \text{Orange}(x) \)
- \( \text{Cat}(x) \)
- \( \text{Fluffy}(x) \)
If you think about it, that's the same as saying that $x$ is an orange and that $x$ is a cat.

<table>
<thead>
<tr>
<th>Available Predicates:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orange($x$)</td>
</tr>
<tr>
<td>Cat($x$)</td>
</tr>
<tr>
<td>Fluffy($x$)</td>
</tr>
</tbody>
</table>

"All $Ps$ are $Qs.$"
\(\forall x. (P(x) \rightarrow Q(x))\)

"Some $Ps$ are $Qs.$"
\(\exists x. (P(x) \land Q(x))\)

"No $Ps$ are $Qs.$"
\(\forall x. (P(x) \rightarrow \neg Q(x))\)

"Some $Ps$ aren't $Qs.$"
\(\exists x. (P(x) \land \neg Q(x))\)

\(\forall x. (x \text{ is orange and } x \text{ is a cat } \rightarrow \text{ Fluffy}(x))\)
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \forall x. (x \text{ is orange and } x \text{ is a cat } \rightarrow \text{Fluffy}(x)) \)

**Available Predicates:**
- Orange(x)
- Cat(x)
- Fluffy(x)

This is something that's a lot easier to translate into first-order logic.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (x \text{ is orange} \land x \text{ is a cat} \rightarrow Fluffy(x)) \]

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)
“All \( P \)s are \( Q \)s.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some \( P \)s are \( Q \)s.”
\[ \exists x. (P(x) \land Q(x)) \]

“No \( P \)s are \( Q \)s.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some \( P \)s aren’t \( Q \)s.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (Orange(x) \land Cat(x) \rightarrow Fluffy(x)) \]

Available Predicates:
- \( Orange(x) \)
- \( Cat(x) \)
- \( Fluffy(x) \)

And, given the predicates we have available, we can translate the left and right halves of that expression directly into first-order logic.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (Orange(x) \land Cat(x) \rightarrow Fluffy(x)) \]
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (Orange(x) \land Cat(x) \rightarrow Fluffy(x)) \]

**Available Predicates:**
- Orange(x)
- Cat(x)
- Fluffy(x)

Although this wasn’t a particularly complicated example, especially compared to what we did in class the other day, I do think it’s helpful to see where it comes from, since we walked through it step-by-step.
"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

Available Predicates:
- Orange(x)
- Cat(x)
- Fluffy(x)

Hopefully that wasn’t too bad! Let’s go and do another one.
Let's change our available set of predicates so that we can talk about whether something's a corgi, whether something's a person, and whether one thing $x$ loves another thing $y$.

**Available Predicates:**
- Corgi($x$)
- Person($x$)
- Loves($x$, $y$)

- "All $P$s are $Q$s."
  $\forall x. (P(x) \rightarrow Q(x))$

- "Some $P$s are $Q$s."
  $\exists x. (P(x) \land Q(x))$

- "No $P$s are $Q$s."
  $\forall x. (P(x) \rightarrow \neg Q(x))$

- "Some $P$s aren't $Q$s."
  $\exists x. (P(x) \land \neg Q(x))$
“All Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. \ (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

“No Ps are Qs.”
\[ \exists x. \ (P(x) \land \neg Q(x)) \]

There's a corgi that loves everyone.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

With these predicates, let's see how to translate this statement into first-order logic.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]
“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]
“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]
“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

There's a corgi that loves everyone.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

Again, we can start off by asking what kind of statement this is. What exactly is it that we’re talking about here?
Fundamentally, we’re saying that somewhere out there in the vast, magical world we live in, there is a corgi that has some specific set of properties.

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)

"All Ps are Qs."
\[ \forall x. \ (P(x) \rightarrow Q(x)) \]

"Some Ps are Qs."
\[ \exists x. \ (P(x) \land Q(x)) \]

"No Ps are Qs."
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. \ (P(x) \land \neg Q(x)) \]

There's a corgi that loves everyone.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

There's a corgi that loves everyone.

Available Predicates:

- \text{Corgi}(x)
- \text{Person}(x)
- \text{Loves}(x, y)

(Specifically, the corgi has the property that it loves everyone!)
“All Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. \ (P(x) \rightarrow \neg Q(x)) \]

“There’s a corgi that loves everyone.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

That statement looks a lot like this one over here – we’re saying that some corgis happen to love everyone.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps aren’t Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (x \text{ is a corgi } \land x \text{ loves everyone}) \]
As before, we'll continue to make incremental progress translating bits and pieces of this formula until we arrive at the final result.

\[ \forall x. (P(x) \rightarrow Q(x)) \]

“All Ps are Qs.”

\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps are Qs.”

\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“No Ps are Qs.”

\[ \exists x. (P(x) \land \neg Q(x)) \]

“Some Ps aren't Qs.”

\[ \exists x. \ (x \text{ is a corgi} \land x \text{ loves everyone}) \]
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“For example, we can directly express the idea that \( x \) is a corgi, so let’s go do that.

Available Predicates:

- Corgi(\( x \))
- Person(\( x \))
- Loves(\( x, y \))

“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“Some Ps aren’t Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \exists x. (\text{Corgi}(x) \land x \text{ loves everyone}) \)
Now, we have to think about how to translate the statement “x loves everyone.”

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)

“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren’t Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (Corgi(x) \land x \text{ loves everyone}) \]
“All Ps are Qs.”
\( \forall x. (P(x) \to Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \to \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \exists x. (\text{Corgi}(x) \land x \text{ loves everyone}) \)

Available Predicates:
- \( \text{Corgi}(x) \)
- \( \text{Person}(x) \)
- \( \text{Loves}(x, y) \)

It’s not immediately clear how to do this given the four general forms we have above. This means that we need to think a bit before we move on.
When translating statements like these, it sometimes helps to introduce variables representing names for things.

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)
<table>
<thead>
<tr>
<th>Statement</th>
<th>Symbol Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;All Ps are Qs.&quot;</td>
<td>$\forall x. (P(x) \rightarrow Q(x))$</td>
</tr>
<tr>
<td>&quot;No Ps are Qs.&quot;</td>
<td>$\forall x. (P(x) \rightarrow \neg Q(x))$</td>
</tr>
<tr>
<td>&quot;Some Ps are Qs.&quot;</td>
<td>$\exists x. (P(x) \land Q(x))$</td>
</tr>
<tr>
<td>&quot;Some Ps aren't Qs.&quot;</td>
<td>$\exists x. (P(x) \land \neg Q(x))$</td>
</tr>
<tr>
<td>&quot;x loves everyone&quot;</td>
<td>$\exists x. (\text{Corgi}(x) \land x \text{ loves every person } y)$</td>
</tr>
</tbody>
</table>

**Available Predicates:**
- Corgi(x)
- Person(x)
- Loves(x, y)

So, for example, we could rewrite "x loves everyone" to "x loves every person y."
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \exists x. (Corgi(x) \land x \text{ loves every person } y) \)

This is suggesting that we’re probably going to want to use one of the templates on the left, since this statement says something about every person y.

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)
Available Predicates:

*Corgi(x)*  
*Person(x)*  
*Loves(x, y)*
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (Corgi(x) \land \text{every person } y \text{ is loved by } x) \]

When I was learning how to write, I remember being told that the passive voice should not be used. But sometimes, like in this case, it’s actually helpful for exposing the structure of what’s going on – every person y is loved by x.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)
“All $P$s are $Q$s.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some $P$s are $Q$s.”
\[ \exists x. (P(x) \land Q(x)) \]

“No $P$s are $Q$s.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some $P$s aren't $Q$s.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (\text{Corgi}(x) \land \text{every person y is loved by x}) \]

Available Predicates:
- Corgi($x$)
- Person($x$)
- Loves($x$, $y$)

If we write things this way, it becomes a bit clearer that this statement matches this first general pattern. Let’s go and apply it!
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren’t Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (\text{Corgi}(x) \land
\forall y. (\text{y is a person} \rightarrow \text{y is loved by } x)) \]

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)
"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (Corgi(x) \land \\
\forall y. (y \text{ is a person } \rightarrow y \text{ is loved by } x) \)
\]

You’ll notice that I’ve written this part of the formula on the next line and indented it. It’s extremely useful to structure the formula this way – it shows what’s nested inside of what and clarifies the scope of the variables involved. While it’s not strictly required that you do this in your own translations, we highly recommend it!

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (\text{Corgi}(x) \land \forall y. (y \text{ is a person } \rightarrow y \text{ is loved by } x)) \]

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

Now that we’re here, we can do the finishing touches of translating this statement by replacing these blue parts with predicates!
"All Ps are Qs."
∀x. (P(x) → Q(x))

"Some Ps are Qs."
∃x. (P(x) ∧ Q(x))

"No Ps are Qs."
∀x. (P(x) → ¬Q(x))

"Some Ps aren't Qs."
∃x. (P(x) ∧ ¬Q(x))

∀x. (Corgi(x) ∧
∀y. (Person(y) → Loves(x, y))
)

Available Predicates:
Corgi(x)
Person(x)
Loves(x, y)

That gives this, our final statement.
"All Ps are Qs."
∀x. (P(x) → Q(x))

"No Ps are Qs."
∀x. (P(x) → ¬Q(x))

"Some Ps are Qs."
∃x. (P(x) ∧ Q(x))

"Some Ps aren't Qs."
∃x. (P(x) ∧ ¬Q(x))

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

And hey! We’re done!
Before we move on, let's pause and look at the formula that we came up with.

\[
\begin{align*}
\forall x. & \quad (P(x) \rightarrow Q(x)) & \text{"All } P\text{s are } Q\text{s."} \\
\forall x. & \quad (P(x) \rightarrow \neg Q(x)) & \text{"No } P\text{s are } Q\text{s."} \\
\exists x. & \quad (P(x) \land Q(x)) & \text{"Some } P\text{s are } Q\text{s."} \\
\exists x. & \quad (P(x) \land \neg Q(x)) & \text{"Some } P\text{s aren't } Q\text{s."} \\
\end{align*}
\]

\[
\exists x. \quad (\text{Corgi}(x) \land \\
\forall y. \quad (\text{Person}(y) \rightarrow \text{Loves}(x, y))
\]

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)
Just as we can use the above patterns to translate the original statement into logic, we can use those same patterns to translate this out of logic and back into English (or any language of your choice, really!)

<table>
<thead>
<tr>
<th>English</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All Ps are Qs.”</td>
<td>$\forall x. (P(x) \rightarrow Q(x))$</td>
</tr>
<tr>
<td>“No Ps are Qs.”</td>
<td>$\forall x. (P(x) \rightarrow \neg Q(x))$</td>
</tr>
<tr>
<td>“Some Ps are Qs.”</td>
<td>$\exists x. (P(x) \land Q(x))$</td>
</tr>
<tr>
<td>“Some Ps aren't Qs.”</td>
<td>$\exists x. (P(x) \land \neg Q(x))$</td>
</tr>
</tbody>
</table>

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\[ \exists x. (Corgi(x) \land \forall y. (Person(y) \rightarrow Loves(x, y))) \]


“All $Ps$ are $Qs$.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some $Ps$ are $Qs$.”
\[ \exists x. (P(x) \land Q(x)) \]

“No $Ps$ are $Qs$.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some $Ps$ aren't $Qs$.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (\text{Corgi}(x) \land \forall y. (\text{Person}(y) \rightarrow \text{Loves}(x, y)) \]

There is a corgi...

Available Predicates:
- $\text{Corgi}(x)$
- $\text{Person}(x)$
- $\text{Loves}(x, y)$

So we can start our translation like this.
"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[
\exists x. (\text{Corgi}(x) \land \forall y. (\text{Person}(y) \rightarrow \text{Loves}(x, y)) \]

There is a corgi...

**Available Predicates:**

- Corgi(x)
- Person(x)
- Loves(x, y)

This part of the statement starts off a statement of the form "all Ps are Qs"...
“All $P$s are $Q$s.”
$\forall x. (P(x) \rightarrow Q(x))$

“No $P$s are $Q$s.”
$\forall x. (P(x) \rightarrow \neg Q(x))$

“Some $P$s are $Q$s.”
$\exists x. (P(x) \land Q(x))$

“Some $P$s aren't $Q$s.”
$\exists x. (P(x) \land \neg Q(x))$

$\exists x. (Corgi(x) \land \\
\forall y. (Person(y) \rightarrow Loves(x, y))$}

There is a corgi
that every person...

Available Predicates:
- Corgi($x$)
- Person($x$)
- Loves($x$, $y$)

...so we can continue our translation like this.
<table>
<thead>
<tr>
<th>“All Ps are Qs.”</th>
<th>“Some Ps are Qs.”</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x. (P(x) \rightarrow Q(x))$</td>
<td>$\exists x. (P(x) \land Q(x))$</td>
</tr>
<tr>
<td>“No Ps are Qs.”</td>
<td>“Some Ps aren't Qs.”</td>
</tr>
<tr>
<td>$\forall x. (P(x) \rightarrow \neg Q(x))$</td>
<td>$\exists x. (P(x) \land \neg Q(x))$</td>
</tr>
</tbody>
</table>

$\exists x. (Corgi(x) \land \forall y. (Person(y) \rightarrow Loves(x, y))$)

There is a corgi that every person is loved by.

**Available Predicates:**
- Corgi(x)
- Person(x)
- Loves(x, y)
We now have a (grammatically awkward) but correct translation of our logic statement back into English.

"All Ps are Qs."
∀x. (P(x) → Q(x))

"Some Ps are Qs."
∃x. (P(x) ∧ Q(x))

"No Ps are Qs."
∀x. (P(x) → ¬Q(x))

"Some Ps aren't Qs."
∃x. (P(x) ∧ ¬Q(x))

∃x. (Corgi(x) ∧
∀y. (Person(y) → Loves(x, y))
)

There is a corgi that every person is loved by.

Available Predicates:
Corgi(x)
Person(x)
Loves(x, y)

We now have a (grammatically awkward) but correct translation of our logic statement back into English.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“All Ps are Qs.”

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“No Ps are Qs.”

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps are Qs.”

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

“Some Ps aren't Qs.”

\[ \exists x. (Corgi(x) \land \forall y. (Person(y) \rightarrow Loves(x, y))) \]

“There is a corgi that loves everyone.”

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

With a bit of English rewriting, we can get back to our original statement. Nifty! Looks like we got it right!
Let's try another translation, just to get some more practice with this skill.

“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“All Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)
"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

**Available Predicates:**
- Corgi(x)
- Person(x)
- Loves(x, y)

---

*Everybody loves at least one corgi.*
Before we walk through this one, why don’t you try translating this one on your own? Try using a similar thought process to the one we used earlier.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

Everybody loves at least one corgi.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. \text{Corgi}(x) \]
\[ \forall x. \text{Person}(x) \]
\[ \forall x. \forall y. \text{Loves}(x, y) \]

Everybody loves at least one corgi.

Did you actually try this? Because if you didn't, you really should. Like, seriously.
<table>
<thead>
<tr>
<th>Statement</th>
<th>Predicate Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All Ps are Qs.”</td>
<td>$\forall x. (P(x) \rightarrow Q(x))$</td>
</tr>
<tr>
<td>“No Ps are Qs.”</td>
<td>$\forall x. (P(x) \rightarrow \neg Q(x))$</td>
</tr>
<tr>
<td>“Some Ps are Qs.”</td>
<td>$\exists x. (P(x) \land Q(x))$</td>
</tr>
<tr>
<td>“Some Ps aren't Qs.”</td>
<td>$\exists x. (P(x) \land \neg Q(x))$</td>
</tr>
</tbody>
</table>

Everybody loves at least one corgi.

Available Predicates:
- \textit{Corgi}(x)
- \textit{Person}(x)
- \textit{Loves}(x, y)

So you translated the statement on your own? Great! Let’s do this one together.
“All Ps are Qs.”
$\forall x. (P(x) \rightarrow Q(x))$

“All Ps are Qs.”
$\forall x. (P(x) \rightarrow \neg Q(x))$

“Some Ps are Qs.”
$\exists x. (P(x) \land Q(x))$

“Some Ps aren't Qs.”
$\exists x. (P(x) \land \neg Q(x))$

Everybody loves at least one corgi.

**Available Predicates:**
- Corgi(x)
- Person(x)
- Loves(x, y)
“All Ps are Qs.”
∀x. (P(x) → Q(x))

“Some Ps are Qs.”
∃x. (P(x) ∧ Q(x))

“All Ps are Qs.”
∀x. (P(x) → Q(x))

“Some Ps aren't Qs.”
∃x. (P(x) ∧ ¬Q(x))

“Some Ps aren't Qs.”
∃x. (P(x) ∧ ¬Q(x))

Everybody loves at least one corgi.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)
"All Ps are Qs."
\( \forall x. (P(x) \rightarrow Q(x)) \)

"No Ps are Qs."
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

"Some Ps are Qs."
\( \exists x. (P(x) \land Q(x)) \)

"Some Ps aren't Qs."
\( \exists x. (P(x) \land \neg Q(x)) \)

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

Everybody loves at least one corgi.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (x \text{ is a person } \rightarrow x \text{ loves at least one corgi}) \]
From here, we can translate the “x is a person” part directly into first-order logic.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

\( \forall x. (Person(x) \rightarrow x \text{ loves at least one corgi}) \)
“All Ps are Qs.”
\( \forall x . (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x . (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x . (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x . (P(x) \land \neg Q(x)) \)

\( \forall x . (Person(x) \rightarrow \)
\( x \ loves \ at \ least \ one \ corgi \)
\)

**Available Predicates:**
- \( \text{Corgi}(x) \)
- \( \text{Person}(x) \)
- \( \text{Loves}(x, y) \)

Now, we have to figure out how to translate that last part.
As before, let's introduce more variables so that we have names for things.

"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (\text{Person}(x) \rightarrow \text{Loves}(x, y)) \]
\[ x \text{ loves at least one corgi } y \]

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

As before, let's introduce more variables so that we have names for things.
<table>
<thead>
<tr>
<th>Statement</th>
<th>Predicate Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>“All Ps are Qs.”</td>
<td>( \forall x. (P(x) \rightarrow Q(x)) )</td>
</tr>
<tr>
<td>“Some Ps are Qs.”</td>
<td>( \exists x. (P(x) \land Q(x)) )</td>
</tr>
<tr>
<td>“No Ps are Qs.”</td>
<td>( \forall x. (P(x) \rightarrow \neg Q(x)) )</td>
</tr>
<tr>
<td>“Some Ps aren’t Qs.”</td>
<td>( \exists x. (P(x) \land \neg Q(x)) )</td>
</tr>
</tbody>
</table>

Available Predicates:

- \( Corgi(x) \)
- \( Person(x) \)
- \( Loves(x, y) \)

And, as before, let’s fiddle around with the verb structure to make clearer what kind of statement this is.
"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (\text{Person}(x) \rightarrow \exists y. \text{Corgi}(y) \land \text{Loves}(x, y)) \]

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

From here it's (hopefully) a bit clearer that this is a "some P's are Q's" statement – some corgis happen to be loved by person x.
"All Ps are Qs."
\( \forall x. \ (P(x) \rightarrow Q(x)) \)

"No Ps are Qs."
\( \forall x. \ (P(x) \rightarrow \neg Q(x)) \)

"Some Ps are Qs."
\( \exists x. \ (P(x) \land Q(x)) \)

"Some Ps aren't Qs."
\( \exists x. \ (P(x) \land \neg Q(x)) \)

\( \forall x. \ (Person(x) \rightarrow \) \\
  \exists y. \ (y \text{ is a corgi} \land y \text{ is loved by } x) \\
) \)

Available Predicates:
- \( Corgi(x) \)
- \( Person(x) \)
- \( Loves(x, y) \)

We can make more progress on our translation by using that template.
“All $P$s are $Q$s.”
$\forall x. (P(x) \rightarrow Q(x))$

“Some $P$s are $Q$s.”
$\exists x. (P(x) \land Q(x))$

“No $P$s are $Q$s.”
$\forall x. (P(x) \rightarrow \neg Q(x))$

“Some $P$s aren't $Q$s.”
$\exists x. (P(x) \land \neg Q(x))$

\[
\forall x. (\text{Person}(x) \rightarrow \\
\exists y. (y \text{ is a corgi } \land y \text{ is loved by } x)
\]
"All Ps are Qs."
\[ \forall x. (P(x) \rightarrow Q(x)) \]

"Some Ps are Qs."
\[ \exists x. (P(x) \land Q(x)) \]

"No Ps are Qs."
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

"Some Ps aren't Qs."
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[
\forall x. (\text{Person}(x) \rightarrow \\
\exists y. (\text{Corgi}(y) \land \text{Loves}(x, y))
\]

...like this! Tada! We're done.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

It's interesting to put the two statements we translated side-by-side with one another.
These statements have a lot of similarities, though they’re clearly different in a number of ways.

**Available Predicates:**
- Corgi(x)
- Person(x)
- Loves(x, y)
One major difference between these two is the order in which the quantifiers appear. The first has them in the order $\exists \forall$, and the second has them in the order $\forall \exists$. 

**Available Predicates:**

- $\text{Corgi}(x)$
- $\text{Person}(x)$
- $\text{Loves}(x, y)$
Something we’d really like to stress is that, when we did these translations, we didn’t just magically “guess” that we needed those particular quantifiers and that they would be in these orders.
"All Ps are Qs."
∀x. (P(x) → Q(x))

"No Ps are Qs."
∀x. (P(x) → ¬Q(x))

"Some Ps are Qs."
∃x. (P(x) ∧ Q(x))

"Some Ps aren't Qs."
∃x. (P(x) ∧ ¬Q(x))

Available Predicates:
Corgi(x)
Person(x)
Loves(x, y)

Instead, we started off with the original statement and incrementally translated it top-down, only adding in the quantifiers when we needed them.
“All Ps are Qs.”
\( \forall x. (P(x) \rightarrow Q(x)) \)

“Some Ps are Qs.”
\( \exists x. (P(x) \land Q(x)) \)

“No Ps are Qs.”
\( \forall x. (P(x) \rightarrow \neg Q(x)) \)

“Some Ps aren't Qs.”
\( \exists x. (P(x) \land \neg Q(x)) \)

Available Predicates:

- Corgi(x)
- Person(x)
- Loves(x, y)
“All Ps are Qs.”
∀x. (P(x) → Q(x))

“Some Ps are Qs.”
∃x. (P(x) ∧ Q(x))

“No Ps are Qs.”
∀x. (P(x) → ¬Q(x))

“Some Ps aren't Qs.”
∃x. (P(x) ∧ ¬Q(x))

Available Predicates:

Corgi(x)
Person(x)
Loves(x, y)

Don’t do that! It’s really, really hard to get right on a first try.
Instead, use the approach we outlined here. Work slowly, going one step at a time, and only adding in quantifiers when you need them.

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (\text{Corgi}(x) \land \forall y. (\text{Person}(y) \rightarrow \text{Loves}(x, y))) \]

\[ \forall x. (\text{Person}(x) \rightarrow \exists y. (\text{Corgi}(y) \land \text{Loves}(x, y))) \]

Available Predicates:
- Corgi(x)
- Person(x)
- Loves(x, y)

If you do, you're a lot less likely to make mistakes.
“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps aren't Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \exists x. (Corgi(x) \land
\forall y. (Person(y) \rightarrow Loves(x, y)) \]

\[ \forall x. (Person(x) \rightarrow
\exists y. (Corgi(y) \land Loves(x, y)) \]

**Available Predicates:**
- Corgi(x)
- Person(x)
- Loves(x, y)

Going back to our programming analogy, you can write a lot of similar programs that all use if statements and for loops.
However, you rarely write programs by just throwing a bunch of loops and if statements randomly and hoping that it’ll work – because chances are, it won’t.
Instead, you work from the outside in – add in a loop when you need it, and if you need to nest an if statement, then you add it when you need it.

**Available Predicates:**
- $\text{Corgi}(x)$
- $\text{Person}(x)$
- $\text{Loves}(x, y)$
So at this point we've gotten some practice with the fundamentals of translation. Pretty much everything else we'll be doing is just more advanced applications of these concepts.
To give you a better sense of how these concepts scale up to more complicated examples, let's walk through some more complex statements and how to translate them. Along the way, you'll see a bunch of nifty tricks and insights that will help you out going forward.
Let's start off by seeing how to talk about pairs of things.
Earlier, we talked about this Java code for iterating over all the elements of an array.

```java
private int sumOf(int[] elems) {
    int result = 0;
    for (int i = 0; i < elems.length; i++) {
        result += elems[i];
    }
    return result;
}
```
Let’s imagine we want to write a different piece of code that iterates over all pairs of elements in the array. How might we do that?

```java
private int sumOf(int[] elems) {
    int result = 0;
    for (int i = 0; i < elems.length; i++) {
        result += elems[i];
    }
    return result;
}
```
Here's one possible option using the venerable "double-for-loop" pattern that you've probably gotten to know and love.
As with the regular "loop over the elements of an array" loop, the double-for-loop is a programming idiom. Once you've seen it enough times, you just know what it means and don't have to think too much about it.
One interesting detail about the double-for-loop pattern is that putting one loop inside of another yields a way of iterating over pairs of things.
Turns out, we can adapt this idea to work in first-order logic as well!
Available Predicates:

Pancake(x)
TasteSimilar(x, y)

Let's imagine that we have these two predicates, one of which says something is a pancake, and one of which says that two things taste similar.
Any two pancakes taste similar

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

How might we translate this statement into first-order logic?
This statement is different from our earlier one because it talks about any possible pair of objects rather than any possible individual object.
The good news is that we can translate it in a way that bears a strong resemblance to the above Java code with a double for loop.

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

Any two pancakes taste similar
private
private void
void printPairsIn(int[]
printPairsIn(int[] elems)
elems) {{
for
for (int
(int ii == 0;
0; ii << elems.length;
elems.length; i++)
i++) {{
for
for (int
(int jj == 0;
0; jj << elems.length;
elems.length; j++)
j++) {{
System.out.println(elems[i]
System.out.println(elems[i] ++ ",
", "" ++ elems[j]);
elems[j]);
}}
}}
}}
Any two pancakes taste similar

Available Predicates:
Pancake(x)
TasteSimilar(x, y)

Specifcally, we'll proceed as follows.


First, let’s introduce some new variables into our English so that we have names for things.

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)

Any two pancakes x and y taste similar
Any pancake $x$ tastes similar to any pancake $y$

Available Predicates:

- Pancake($x$)
- TasteSimilar($x$, $y$)

We can then rejigger the English statement so that it looks like this. After all, this means the same thing as what we started with.
Now, we can think back to our Aristotelean form templates that we just got really familiar with and see how to apply them here.

Any pancake x tastes similar to any pancake y

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)
Since this statement says something to the effect of "any pancake x has some special property..."
... we can begin translating it into logic like this.

\[ \forall x. (Pancake(x) \rightarrow x \text{ tastes similar to any pancake } y) \]
Now, let's look at that middle portion and see if we can translate it as well.

Available Predicates:

Pancake(x)
TasteSimilar(x, y)

∀x. (Pancake(x) →
  x tastes similar to any pancake y)
  )
Reordering the statement gives us this to work with, which exposes a bit more structure.

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
We can then rewrite it like this.

\[
\forall x. (Pancake(x) \rightarrow \\
\forall y. (Pancake(y) \rightarrow \\
\quad x \text{ tastes similar to } y)
\]

**Available Predicates:**

- Pancake(x)
- TasteSimilar(x, y)
As a final step, we'll translate that innermost portion.

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)
private void printPairsIn(int[] elems) {
    for (int i = 0; i < elems.length; i++) {
        for (int j = 0; j < elems.length; j++) {
            System.out.println(elems[i] + " , " + elems[j]);
        }
    }
}

∀x. (Pancake(x) →
    ∀y. (Pancake(y) →
        TasteSimilar(x, y)
    )
)

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)
We now have a statement that says that any two pancakes taste similar. (We can debate whether this is true or not in a separate guide.)

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
Hopefully, you can notice that there's a bit of a parallel to the Java double for loop given above.

Available Predicates:

Pancake(x)
TasteSimilar(x, y)
If you think of quantifiers as a sort of “loop over everything” – which isn’t that far from the truth – then the program and the formula both say “loop over one thing, then loop over another, then do something with the pair.”

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
So if you ever need to write something where you’re dealing with a pair of things, you now know how! You can just write two independent quantifiers like this.

\[
\forall x. (\text{Pancake}(x) \to
\forall y. (\text{Pancake}(y) \to
\text{TasteSimilar}(x, y))
\)\]

Available Predicates:
- \(\text{Pancake}(x)\)
- \(\text{TasteSimilar}(x, y)\)
It turns out, though, that there’s another way to express this concept that some people find a bit easier to wrap their head around. For completeness, let’s quickly talk about this before moving on.

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
Let's go back to our original statement.

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

Any two pancakes taste similar
As before, let's add in some variables names so that we have ways of keeping our pancakes straight. (Ever gotten your pancakes confused? It's a horrible way to start off your day.)

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

Any two pancakes x and y taste similar
The idea is that we know that, at this point, we’re going to be reasoning about a pair of pancakes, and we’re going to reason about them right now.

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

Any two pancakes x and y taste similar
Therefore, rather than introducing two quantifiers at different points in time, we'll introduce both quantifiers at the same time.

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

Any two pancakes x and y taste similar
∀x. ∀y. (x and y are pancakes → x and y taste similar)

Available Predicates:
- Pancake(x)
- TasteSimilar(x, y)

...like this.
Generally speaking, it is not a good idea to introduce quantifiers for variables all at once, but in the special case of working with pairs, it’s perfectly safe.

Available Predicates:

\[
\forall x. \forall y. (x \text{ and } y \text{ are pancakes } \rightarrow \\
x \text{ and } y \text{ taste similar})
\]
∀x. ∀y. (x and y are pancakes → x and y taste similar)

Available Predicates:

Pancake(x)
TasteSimilar(x, y)

So now all we have to do is translate each of the remaining English parts into English.
Here’s one way to do this.

Available Predicates:
- \( \text{Pancake}(x) \)
- \( \text{TasteSimilar}(x, y) \)
And we're done! This is a totally valid way to translate our original statement into first-order logic.

\[ \forall x. \forall y. (\text{Pancake}(x) \land \text{Pancake}(y) \rightarrow \text{TasteSimilar}(x, y)) \]
Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)

It's interesting, and useful, to put this second translation side-by-side with our original one.
These statements look pretty different, but they say exactly the same thing. Both are perfectly correct.

Available Predicates:

Pancake(x)
TasteSimilar(x, y)
There's actually something pretty cool and pretty deep going on here.

Available Predicates:

Pancake(x)
TasteSimilar(x, y)
Available Predicates:

- Pancake(\(x\))
- TasteSimilar(\(x, y\))

For now, ignore the quantifiers. Just look at the predicates and how they relate.
Abstractly, here are the two propositional logic patterns used in the two statements.

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
These statements are actually logically equivalent to one another. (If you've checked out the Guide to Negating Formulas, you'll see a cool way to derive this!)

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
This pattern – changing a chain of implications into a single implication and a lot of ANDs and vice-versa – is sometimes called Currying and has applications in functional programming. (This is a total aside... you're not expected to know this.)

Available Predicates:

Pancake(x)
TasteSimilar(x, y)
Ultimately, what’s important is that you understand that both of these statements say exactly the same thing and that you end up comfortable working with both of them. Feel free to use whichever one you like more, but make sure you can quickly interpret both.

Available Predicates:

- Pancake(x)
- TasteSimilar(x, y)
Let's do another example of where we might want to go and work with pairs.
Let's switch our predicates from pancakes to people.

Available Predicates:

\[ \text{Person}(x) \]
\[ \text{Knows}(x, y) \]
How might we translate this statement into first-order logic?

Everyone knows at least two people

Available Predicates:

Person(x)
Knows(x, y)
Everyone knows at least two people

Available Predicates:

- Person(x)
- Knows(x, y)

Well, it seems like there’s going to be a pair involved here somewhere, since there’s something about “at least two people” here.
Everyone knows *at least two people*

However, that **does not** mean that we should immediately start writing out something about a pair of people. Remember – we should only introduce quantifiers when we immediately need them, and it’s not clear that we need to start talking about these two people yet.

Available Predicates:

- \textit{Person}(x)
- \textit{Knows}(x, y)
Instead, let's look at the overall structure of this statement and see what it is that we're trying to say.

Everyone knows at least two people

Available Predicates:
- Person(x)
- Knows(x, y)
As usual, let's start by introducing some variables so that we can keep track of who we're talking about.

**Available Predicates:**
- Person(x)
- Knows(x, y)

Every person x knows at least two people y and z
∀x. (Person(x) →
    x knows at least two people y and z
)

Available Predicates:
- Person(x)
- Knows(x, y)

We can then partially translate this statement using the techniques we've seen so far.
∀x. (Person(x) → x knows at least two people y and z)

Available Predicates:

Person(x)
Knows(x, y)
There are a couple of ways to do it, and since we’ve got time, we’ll do it in two different ways.

Available Predicates:

- Person(x)
- Knows(x, y)

∀x. (Person(x) →
   x knows at least two people y and z)

}
Previously, we talked about working with pairs in a universally-quantified setting. Here, though, this particular pair is going to be existentially quantified, since we’re saying that there exist two people with certain properties.

\[
\forall x. \ (\text{Person}(x) \rightarrow \ x \ \text{knows at least two people} \ y \ \text{and} \ z)
\]

Available Predicates:
- Person(x)
- Knows(x, y)
∀x. (Person(x) →
\text{there are two people } y \text{ and } z \text{ that } x \text{ knows}
)

\text{Available Predicates:}
- Person(x)
- Knows(x, y)

It might be easier to see that if we rewrite things like this.

Thinking back to our double for loop intuition, let's see if we can translate this statement by nesting some existential statements inside of one another.

Available Predicates:

- Person(x)
- Knows(x, y)

∀x. (Person(x) → there are two people y and z that x knows)

.
∀x. (Person(x) → there is a person y that x knows and a different person z that x knows.)
We can now make some progress translating this.

Available Predicates:

- Person(x)
- Knows(x, y)

∀x. (Person(x) →
     ∃y. (Person(y) ∧ Knows(x, y) ∧
         there is a different person z that x knows)
     )

We can now make some progress translating this.
We can then finish up the rest of this translation by translating this blue part in the middle. But that shouldn’t be too bad!

Available Predicates:
- Person(x)
- Knows(x, y)
Here's one way to do it.

**Available Predicates:**

- `Person(x)`
- `Knows(x, y)`

\[
\forall x. (\text{Person}(x) \rightarrow \\
\exists y. (\text{Person}(y) \land \text{Knows}(x, y) \land \\
\exists z. (\text{Person}(z) \land \text{Knows}(x, z) \land \\
\quad z \text{ is a different person from } y)
\) )
) )
\]
The last step is to say that \( z \) and \( y \) aren't the same person.

\[
\forall x. \ (\text{Person}(x) \rightarrow \\
\exists y. \ (\text{Person}(y) \land \text{Knows}(x, y) \land \\
\exists z. \ (\text{Person}(z) \land \text{Knows}(x, z) \land \\
z \text{ is a different person from } y
\)
)
)
\]

\textbf{Available Predicates:}

\textit{Person}(x) \\
\textit{Knows}(x, y)
Even though we didn’t explicitly list it in our list of predicates, remember that first-order logic has the equality predicate built into it, so we’re always allowed to state that two things are the same or are different.

Available Predicates:

Person(x)
Knows(x, y)
Here’s one way to do that.

Available Predicates:

- Person(x)
- Knows(x, y)
And hey! We're done!

Available Predicates:

- Person(x)
- Knows(x, y)
\[ \forall x. (\text{Person}(x) \rightarrow \\
\exists y. (\text{Person}(y) \land \text{Knows}(x, y) \land \\
\exists z. (\text{Person}(z) \land \text{Knows}(x, z) \land z \neq y) \\
\) \)
\]

Notice how we’re using a pair of nested existential quantifiers to express the idea that there’s a pair of people with specific properties.

**Available Predicates:**
- **Person(x)**
- **Knows(x, y)**
Hopefully, this seems familiar, since it's closely related to the analogous doubly-nested quantifiers we saw when talking about pairs of pancakes.

Available Predicates:

- Person(x)
- Knows(x, y)
Just as we could write “any pair of pancakes” in two ways, we can write “some pair of different people” in two ways.

Available Predicates:

- Person(x)
- Knows(x, y)
Here's the alternative approach. Here, we introduce the quantifiers for $y$ and $z$ at the same time, then constrain $y$ and $z$ with preconditions at the same time.

**Available Predicates:**

- $\text{Person}(x)$
- $\text{Knows}(x, y)$

\[
\forall x. (\text{Person}(x) \rightarrow \\
\exists y. (\text{Person}(y) \land \text{Knows}(x, y) \land \\
\exists z. (\text{Person}(z) \land \text{Knows}(x, z) \land z \neq y))
\]

\[
\forall x. (\text{Person}(x) \rightarrow \\
\exists y. \exists z. (\text{Person}(y) \land \text{Person}(z) \land z \neq y \land \\
\text{Knows}(x, y) \land \text{Knows}(x, z))
\]
These two approaches are completely equivalent, and both of them are correct. As with quantifying over pairs using $\forall$, it's a good idea to get comfortable with quantifying over pairs using $\exists$ with both of these approaches.

Available Predicates:
- Person(x)
- Knows(x, y)
On Problem Set Two, you’ll get to consider a variation on this problem: how would you express the idea that this person $x$ knows exactly two people? That’s a trickier proposition, but (hypothetically speaking) you may want to use this basic setup as a starting point.

Available Predicates:

- `Person(x)`
- `Knows(x, y)`

$$\forall x. \ (\text{Person}(x) \rightarrow \\
\exists y. \ (\text{Person}(y) \land \text{Knows}(x, y) \land \\
\exists z. \ (\text{Person}(z) \land \text{Knows}(x, z) \land z \neq y) \\
)$$

$$\forall x. \ (\text{Person}(x) \rightarrow \\
\exists y. \ \exists z. \ (\text{Person}(y) \land \text{Person}(z) \land z \neq y \land \\
\text{Knows}(x, y) \land \text{Knows}(x, z) \\
)$$
There's one last topic I'd like to speak about in this guide, and that's what happens when you start talking about sets and set theory in first-order logic.
Even if you don’t find yourself talking about set theory much in first-order logic, the lessons we’ll learn in the course of exploring these sorts of translations are extremely valuable, especially when it comes to checking your work.
Let's imagine that we have the set of predicates over to the left. We can say that something is a set, that one thing is an element of something else, that something is an integer, and that something is negative.

Available Predicates:

- \(\text{Set}(x)\)
- \(x \in y\)
- \(\text{Integer}(x)\)
- \(\text{Negative}(x)\)
How might we translate this statement into first-order logic?

The set of all natural numbers exists

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)
The set of all natural numbers exists

Available Predicates:
- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$

This statement is, in many ways, quite different from the ones we've seen so far.
First, the statement doesn’t seem to look anything like the Aristotelian forms that we saw earlier. Instead, it just says that something exists.

**Available Predicates:**

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
Second, this statement refers to a specific thing – the set of all natural numbers – and so it's not exactly clear how we'd actually translate this into logic.

Available Predicates:

Set(x)  
x ∈ y  
Integer(x)  
Negative(x)
If you encounter a statement like this one, which asks you to show that something exists, it often helps to reframe the statement to translate in a different light.

**Available Predicates:**

- Set(x)
- $x \in y$
- Integer(x)
- Negative(x)

*The set of all natural numbers exists*
Rather than saying “this specific thing exists.”

Available Predicates:
- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)

The set of all natural numbers exists
There is a set that is the set of all natural numbers
There is a set that is the set of all natural numbers

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)
Here's one way that we can get this translation started.

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)

$\exists S. (Set(S) \land S$ is the set of all natural numbers $)$
So now we need to find a way to pin down the fact that $S$ is the set of all natural numbers.

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
To do so, let’s take a few minutes to think about how we might do that.

Available Predicates:

- Set(x)
- \( x \in y \)
- Integer(x)
- Negative(x)
If we're going to say that $S$ is the set of all natural numbers, we're probably going to need to find some way to talk about its elements. After all, sets are uniquely defined by their elements, so if we want to say that we have a set with a certain property, we can do so by saying that it has the right elements.

Available Predicates:

- $\exists S. (\text{Set}(S) \land \text{S is the set of all natural numbers})$
- $\text{Set}(x)$
- $x \in y$
- $\text{Integer}(x)$
- $\text{Negative}(x)$
We're not sure how we're going to do that, but at least we know to keep an eye out for that.

\[ \exists S. \ ( \text{Set}(S) \land S \text{ is the set of all natural numbers} ) \]

**Available Predicates:**
- \( \text{Set}(x) \)
- \( x \in y \)
- \( \text{Integer}(x) \)
- \( \text{Negative}(x) \)

We're not sure how we're going to do that, but at least we know to keep an eye out for that.
Next, we need to find a way to say that something is a natural number.

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$

$\exists S. \ (Set(S) \land S \text{ is the set of all natural numbers}$

)
We have the ability to say that something is an integer or that something is negative, and that might come in handy - the natural numbers are the integers that aren't negative!

\[ \exists S. (\text{Set}(S) \land S \text{ is the set of all natural numbers}) \]
So even if we have no idea where we’re going right now, we at least know that (1) we want to say something about the elements of $S$, and (2) we’re going to try to say something about how they’re integers that aren’t negative.

Available Predicates:
- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
Rather than just show you the final answer, let's see how not to do this.

\[ \exists S. (\text{Set}(S) \land S \text{ is the set of all natural numbers}) \]

Available Predicates:

- \( \text{Set}(x) \)
- \( x \in y \)
- \( \text{Integer}(x) \)
- \( \text{Negative}(x) \)
As before, I'm going to put up the emergency warning flags indicating that we're doing something wrong here.

Available Predicates:

Set(x)
x ∈ y
Integer(x)
Negative(x)

∃S. (Set(S) ∧ S is the set of all natural numbers)
Let's try an initial approach. What does it mean for $S$ to be the set of all natural numbers?

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$

$\exists S. (Set(S) \land S \text{ is the set of all natural numbers})$
Here’s a reasonable – but incorrect – way of thinking about it. If you don’t see why this is incorrect, don’t worry! It’s subtle, which is precisely why we’re taking the time to go down this route.

Available Predicates:

\[
\exists S. (Set(S) \land S \text{ contains all the natural numbers})
\]
∃S. (Set(S) ∧
    S contains all the natural numbers)

Available Predicates:
- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)

Now, how might we translate this red statement into first-order logic?
Again, let's change up the ordering of the English to expose a bit more structure.

Available Predicates:

- \( \exists S. (\text{Set}(S) \land \text{every natural number is an element of } S) \)
This matches one of our nice Aristotelian forms, so we can rewrite it like this.

\[ \exists S. (\text{Set}(S) \land \forall x. (x \text{ is a natural number } \rightarrow x \text{ is an element of } S) \) \]
We can clean up the consequent of that implication (the part that's implied) using the predicates we have available.

Available Predicates:

- Set(x)
- \( x \in y \)
- Integer(x)
- Negative(x)
As for the antecedent - as we saw earlier, the natural numbers are the integers that aren't negative, so we can say something like this:

$$\exists S. \ (\text{Set}(S) \land \forall x. \ (x \text{ is an integer and } x \text{ isn't negative} \rightarrow x \in S)$$
We can then translate that into logic like this. Done! ...ish

\[
\exists S. (\text{Set}(S) \land \\
\forall x. (\text{Integer}(x) \land \neg \text{Negative}(x) \rightarrow \\
\quad x \in S)
\]

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)
So it seems like we're done, but we still have those big red warning signs everywhere. Why doesn't this work?

Available Predicates:

- $\text{Set}(x)$
- $x \in y$
- $\text{Integer}(x)$
- $\text{Negative}(x)$
Well, fundamentally, the way this statement works is by saying “there is some set $S$ that is the set of all natural numbers.”

Available Predicates:

- Set($x$)
- $x \in y$
- Integer($x$)
- Negative($x$)
Since this is an existentially-quantified statement, it's true if we can find a choice of $S$ that makes it true.

Available Predicates:

- Set($x$)
- $x \in y$
- Integer($x$)
- Negative($x$)
We've tried to structure this statement with the intent that, specifically, the only choice of \( S \) that will work should be \( \mathbb{N} \), the set of all natural numbers.

Available Predicates:
- \( Set(x) \)
- \( x \in y \)
- \( Integer(x) \)
- \( Negative(x) \)
If we can make this statement true without choosing $S$ to be the set of all natural numbers, then we haven't actually stated that $\mathbb{N}$ exists.

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
Unfortunately, it is entirely possible to choose a set besides $\mathbb{N}$ that makes this formula true.

Available Predicates:

- Set(x)
- $x \in y$
- Integer(x)
- Negative(x)
Specifically, what if we choose $S$ to be the set $\mathbb{R}$?

$$\exists S. \ (\text{Set}(S) \land \forall x. \ (\text{Integer}(x) \land \neg \text{Negative}(x) \rightarrow x \in S)$$

Choose $S = \mathbb{R}$.
That means that $S$ is definitely a set.

\[
\exists S. \ (Set(S) \land \\
\forall x. \ (Integer(x) \land \neg Negative(x) \rightarrow x \in S)
\]

**Choose** $S = \mathbb{R}$.

*Available Predicates:*

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
\( \exists S. (\text{Set}(S) \land \\
\quad \forall x. (\text{Integer}(x) \land \neg \text{Negative}(x) \rightarrow x \in S) \)
\)

Choose \( S = \mathbb{R} \).

Available Predicates:
- \( \text{Set}(x) \)
- \( x \in y \)
- \( \text{Integer}(x) \)
- \( \text{Negative}(x) \)

...and this part of the formula is true: every nonnegative integer is contained in \( S \).
This means that the statement we’ve written doesn’t say “the set of all natural numbers exists.” It says “there is some set that contains all the natural numbers,” which is similar, but not the same thing.

Choose $S = \mathbb{R}$. 

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
Fundamentally, the issue with this translation is that we've put on a set of minimum requirements on $S$, not a set of exact requirements. As a result, it's possible to make this formula true with a choice of $S$ that has some, but not all, of the properties of $\mathbb{N}$. We're going to need to rework the formula to correct that deficiency.

Choose $S = \mathbb{R}$.

**Available Predicates:**

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
To do so, let’s go back in time to the last point where everything was working correctly...

Available Predicates:

Set(x)
\( x \in y \)
Integer(x)
Negative(x)

\[ \exists S. (Set(S) \land \forall x. (\text{Integer}(x) \land \neg\text{Negative}(x) \rightarrow \ x \in S) ) \]

Choose \( S = \mathbb{R} \).
\[ \exists S. (Set(S) \land S \text{ is the set of all natural numbers}) \]
\[ \exists S. (\text{Set}(S) \land S \text{ is the set of all natural numbers}) \]
So what other approaches can we take?

\[ \exists S. (\text{Set}(S) \land S \text{ is the set of all natural numbers}) \]

**Available Predicates:**

- `Set(x)`
- `x \in y`
- `Integer(x)`
- `Negative(x)`

So what other approaches can we take?
I'm going to show you another approach that doesn't work, which is a common strategy that we see students take after they realize that the previous approach is incorrect.

\[ \exists S. (\text{Set}(S) \land \text{S is the set of all natural numbers}) \]

**Available Predicates:**

- Set(x)
- \( x \in y \)
- Integer(x)
- Negative(x)
\[ \exists S. (\text{Set}(S) \land \text{S is the set of all natural numbers}) \]
Maybe we should think about this differently. The reason that we could get away with choosing $\mathbb{R}$ for our set $S$ was that our formula said "$S$ has to have at least these elements." What if we try a different tactic and say that $S$ has to have at most these elements?

Available Predicates:
- $\exists S. (\text{Set}(S) \land S \text{ is the set of all natural numbers})$
\[ \exists S. \ (\text{Set}(S) \land \text{the only elements of } S \text{ are natural numbers}) \]

Available Predicates:

- \( \text{Set}(x) \)
- \( x \in y \)
- \( \text{Integer}(x) \)
- \( \text{Negative}(x) \)

That is, what if we try replacing the previous blue statement with this red statement?
Available Predicates:

\[
\begin{align*}
&\text{Set}(x) \\
&x \in y \\
&\text{Integer}(x) \\
&\text{Negative}(x)
\end{align*}
\]

This isn't the same thing as before... do you see why?
Given that it’s different, let’s see if we can translate this into first-order logic.

Available Predicates:
- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
Rewording this statement and introducing some variables helps make clearer what we're going to do next.

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)

∃S. (Set(S) ∧ every element of S is a natural number)
\[ \exists S. (\text{Set}(S) \land \forall x. (x \in S \rightarrow x \text{ is a natural number}) \) \)
And, since we've seen earlier how to express the idea that \( x \) is a natural number...

Available Predicates:

- \( Set(x) \)
- \( x \in y \)
- \( Integer(x) \)
- \( Negative(x) \)
Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)

...we can complete our translation like this.

\[ \exists S. (Set(S) \land \forall x. (x \in S \rightarrow Integer(x) \land \neg \text{Negative}(x)) \]
∃S. (Set(S) \land 
\forall x. (x \in S \rightarrow 
\text{Integer}(x) \land \neg\text{Negative}(x))
)
As before, we should check to make sure that the only way this statement can be made true is by picking \( S \) to be the set of all natural numbers. Is that really the case?

Available Predicates:
- \( Set(x) \)
- \( x \in y \)
- \( Integer(x) \)
- \( Negative(x) \)
\[ \exists S. (Set(S) \land \forall x. (x \in S \rightarrow \text{Integer}(x) \land \neg \text{Negative}(x)) \) \]

Choose \( S = \{137\} \).

Available Predicates:

- Set(x)
- \( x \in y \)
- Integer(x)
- Negative(x)

Unfortunately, no. What if we pick this choice for \( S \)?
\( \exists S. (\text{Set}(S) \land \forall x. (x \in S \rightarrow \text{Integer}(x) \land \neg \text{Negative}(x))) \)

Choose \( S = \{137\} \).
\[
\exists S. \ (\text{Set}(S) \land \\
\forall x. \ (x \in S \rightarrow \\
\text{Integer}(x) \land \neg \text{Negative}(x))
\]

Choose \( S = \{137\} \).

... and this statement is true: every element of \( S \) is indeed a natural number.

Available Predicates:
- \( \text{Set}(x) \)
- \( x \in y \)
- \( \text{Integer}(x) \)
- \( \text{Negative}(x) \)
∃S. (Set(S) \land \\
\forall x. (x \in S \rightarrow \\
\quad \text{Integer}(x) \land \neg \text{Negative}(x) \\
\quad ) \\
\quad ) \\

Choose S = \{137\}.

\textbf{Available Predicates:}

- Set(x)
- x \in y
- Integer(x)
- Negative(x)

So our translation isn’t correct – even if there is no set of all natural numbers, we can still make the formula true by picking some other set... in this case, any set that happens to only contain natural numbers.
∃S. (Set(S) ∧
    ∀x. (x ∈ S →
       Integer(x) ∧ ¬Negative(x)
    )
  )
)

Choose $S = \emptyset$

Available Predicates:

- Set(x)
- $x \in y$
- Integer(x)
- Negative(x)

Interesting, we could have also chosen $S = \emptyset$ as a counterexample. Then this inner statement happens to be vacuously true because there are no elements of $S$ to speak of!
So here are our two attempted translations, each of which isn't correct.

Available Predicates:

- `Set(x)`
- `x ∈ y`
- `Integer(x)`
- `Negative(x)`
Interestingly, although each of them is wrong, they're wrong in complementary ways.

Available Predicates:

- Set(x)
- $x \in y$
- Integer(x)
- Negative(x)
Our first statement was wrong because it let us choose sets that had all the natural numbers, plus some other things that shouldn’t be there.

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
However, notice that we can't pick an $S$ that misses any natural numbers, because the inside says that all the natural numbers should be there.

Available Predicates:

$Set(x)$
$x \in y$
$Integer(x)$
$Negative(x)$
This second statement was incorrect because it let us choose sets $S$ with too few elements, since all it required was that elements that were present were natural numbers.

Available Predicates:

- $\text{Set}(x)$
- $x \in y$
- $\text{Integer}(x)$
- $\text{Negative}(x)$
However, note that this formula doesn’t let us choose a set $S$ that contains anything that’s not a natural number, since it requires everything in $S$ to be a natural number.

Available Predicates:

- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
In a sense, you can think of our translations this way...

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)
This first part says \( \mathbb{N} \subseteq S \), since it requires that every natural number be in \( S \).

Available Predicates:

- \( \text{Set}(x) \)
- \( x \in y \)
- \( \text{Integer}(x) \)
- \( \text{Negative}(x) \)
This second part says $S \subseteq \mathbb{N}$, since it requires that every element of $S$ be a natural number.

Available Predicates:

$Set(x)$  
$x \in y$  
$Integer(x)$  
$Negative(x)$
In other words, each individual constraint doesn’t guarantee that $S$ has to be $\mathbb{N}$, but the two statements collectively would require that $S = \mathbb{N}$!
Let’s wind back the clock and see if we can use this to our advantage.

Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)
\[ \exists S. (\text{Set}(S) \land \text{S is the set of all natural numbers}) \]
The problem was that in the last two cases, we kept mistranslating this blue statement, which got us the wrong answer.

Available Predicates:

\[ \exists S. (\text{Set}(S) \land S \text{ is the set of all natural numbers}) \]
∀S. (Set(S) ∧
    S ⊆ ℕ ∧
    ℕ ⊆ S
)
We can then snap in the two parts of the formulas that we built up earlier...

Available Predicates:

\-
Set(x)
\-
\ x \in y
\-
\text{Integer}(x)
\-
\text{Negative}(x)
\[ \exists S. (\text{Set}(S) \land \\
\forall x. (x \in S \rightarrow \\
\text{Integer}(x) \land \lnot \text{Negative}(x)) \\
\land \\
\forall x. (\text{Integer}(x) \land \lnot \text{Negative}(x) \rightarrow \\
x \in S) \) \]
And hey! This actually works!

\[ \exists S. \ (\text{Set}(S) \land \forall x. \ (x \in S \rightarrow (\text{Integer}(x) \land \neg \text{Negative}(x)) ) \land \forall x. \ (\text{Integer}(x) \land \neg \text{Negative}(x) \rightarrow x \in S) ) \]

Available Predicates:
- \text{Set}(x)
- x \in y
- \text{Integer}(x)
- \text{Negative}(x)
If we choose an $S$ that contains something it shouldn’t, this part will catch it…

Available Predicates:
- $Set(x)$
- $x \in y$
- $Integer(x)$
- $Negative(x)$
\[ \exists S. (\text{Set}(S) \land \\
    \forall x. (x \in S \rightarrow \\
    \text{Integer}(x) \land \neg \text{Negative}(x)) \land \\
    \forall x. (\text{Integer}(x) \land \neg \text{Negative}(x) \rightarrow \\
    x \in S) \) \]
So in that sense, we have a working formula!

\[ \exists S. (\text{Set}(S) \land \\
\forall x. (x \in S \to \\
\text{Integer}(x) \land \neg \text{Negative}(x) \\
) \land \\
\forall x. (\text{Integer}(x) \land \neg \text{Negative}(x) \to \\
\quad x \in S \\
) ) \]

Available Predicates:
- \text{Set}(x)
- x \in y
- \text{Integer}(x)
- \text{Negative}(x)
As a final step, though, we can clean this up a bit.

∀x. (x ∈ S → (Set(S) ∧ ¬Negative(x) → x ∈ S)) ∧ (∃S. (Set(S) ∧ ∀x. (x ∈ S → Integer(x) ∧ ¬Negative(x))))
Look at these two implications.
Notice anything about them?

\[ \exists S. \ (Set(S) \land \\
\forall x. \ (x \in S \rightarrow \\
\ \ \ \ \ \ \ \ \Integer(x) \land \neg \Negative(x) \\
\ ) \land \\
\forall x. \ (\Integer(x) \land \neg \Negative(x) \rightarrow \\
\ \ \ \ \ \ \ \ x \in S \\
\) \]
Available Predicates:

- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)

Except for the fact that the antecedent and the consequent have been swapped, they're the same!
\[\exists S. (\text{Set}(S) \land \forall x. (x \in S \rightarrow \text{Integer}(x) \land \neg \text{Negative}(x)) \land \forall x. (\text{Integer}(x) \land \neg \text{Negative}(x) \rightarrow x \in S) )\]

Available Predicates:
- \text{Set}(x)
- x \in y
- \text{Integer}(x)
- \text{Negative}(x)

And hey... don't we have a special symbol to say that \(A \rightarrow B\) and that \(B \rightarrow A\)?
So as a final step, let's take this formula and rewrite it using the biconditional connective.

Available Predicates:
- Set(x)
- x ∈ y
- Integer(x)
- Negative(x)
\[ \exists S. \ (Set(S) \land \forall x. \ (x \in S \leftrightarrow \text{Integer}(x) \land \neg \text{Negative}(x)) \) \]
\begin{align*}
\exists S. \ (Set(S) \land \\
\forall x. \ (x \in S \leftrightarrow Integer(x) \land \neg Negative(x)))
\end{align*}

Available Predicates:

- Set(x)
- \(x \in y\)
- Integer(x)
- Negative(x)

This single biconditional contains everything we need.
In the forwards direction, it says “everything in S needs to be a natural number.”

Available Predicates:

- Set(x)
- \( x \in y \)
- Integer(x)
- Negative(x)
\[ \exists S. (Set(S) \land \forall x. (x \in S \leftrightarrow Integer(x) \land \neg Negative(x))) \]

**Available Predicates:**

- Set(x)
- x \in y
- Integer(x)
- Negative(x)

In the reverse direction, it says “every natural number needs to be in S.”
Generally, if you’re trying to write a statement in first-order logic that says that some set exists (which, hypothetically speaking, might happen sometime soon), you might find yourself using a biconditional to pin down the elements of the set. It’s an easy way to say “the set contains precisely these elements.”

Available Predicates:

- Set(x)
- \( x \in y \)
- Integer(x)
- Negative(x)
Wow! We’ve covered a ton in this guide. Before we wrap up, let’s briefly recap the major themes and ideas from what we’ve seen here.
“All $P$s are $Q$s.”
$\forall x. \ (P(x) \rightarrow Q(x))$

“Some $P$s are $Q$s.”
$\exists x. \ (P(x) \land Q(x))$

“No $P$s are $Q$s.”
$\forall x. \ (P(x) \rightarrow \neg Q(x))$

“Some $P$s aren't $Q$s.”
$\exists x. \ (P(x) \land \neg Q(x))$

First, we saw these four basic statement building blocks. These are idiomatic expressions in first-order logic – the same way that a for loop over an array is idiomatic in most programming languages – and are extremely useful in assembling more complex statements.
We saw that translating things incrementally, going one step at a time and judiciously rewriting the English, is a reliable way to end up with good translations. Plus, it sidesteps a ton of classes of mistakes.

“All Ps are Qs.”
\[ \forall x. (P(x) \rightarrow Q(x)) \]

“No Ps are Qs.”
\[ \forall x. (P(x) \rightarrow \neg Q(x)) \]

“Some Ps are Qs.”
\[ \exists x. (P(x) \land Q(x)) \]

“Some Ps aren’t Qs.”
\[ \exists x. (P(x) \land \neg Q(x)) \]

\[ \forall x. (Person(x) \rightarrow \neg \neg Q(y)) \]
\[ x \text{ loves at least one corgi } y \]
∀x. (Pancake(x) →
    ∀y. (Pancake(y) →
        TasteSimilar(x, y)
    )
}

∀x. ∀y. (Pancake(x) ∧ Pancake(y) →
    TasteSimilar(x, y)
)

We saw how to quantify over pairs of things, and saw that there are multiple ways of doing so.
We saw that we can check our work by plugging in specific values and seeing whether they work the way we expect them to work.

\[ \exists S. (\text{Set}(S) \land \forall x. (x \in S \rightarrow \text{Integer}(x) \land \neg \text{Negative}(x)) \) \]

Choose \( S = \{137\} \).

We saw that we can check our work by plugging in specific values and seeing whether they work the way we expect them to work.
And, finally, we saw where biconditionals come from, especially in set theory contexts.

\[ \exists S. \ (\text{Set}(S) \land \forall x. \ (x \in S \leftrightarrow \text{Integer}(x) \land \neg \text{Negative}(x))) \]
Hope this helps!

Please feel free to ask questions if you have them.
Did you find this useful? If so, let us know! We can go and make more guides like these.