Math Time!!!

"and" / "or"

"and" / "or"

```
Logical "AND". Called a "conjunct". Written in ASCII as /\
Written in Python as and. Written in C as &&

TRUE /\ TRUE = TRUE

TRUE /\ FALSE = FALSE

FALSE /\ FALSE = FALSE

FALSE /\ TRUE = FALSE
```

"and" / "or"

```
Logical "AND". Called a "conjunct". Written in ASCII as /\
 Written in Python as and. Written in C as &&
  TRUE /\ TRUE
                    = TRUE
  TRUE /\ FALSE = FALSE
  FALSE /\ FALSE = FALSE
  FALSE /\ TRUE = FALSE
Logical "OR". Called a "disjunct". Written in ASCII as \/
Written in Python as or. Written in C as | |
 TRUE \/ TRUE
                   = TRUE
 TRUE \/ FALSE = TRUE
 FALSE \/ FALSE = FALSE
```

= TRUE

FALSE \/ TRUE

means "equality". It is NOT an assignment operator. It is a boolean operator. It is the = you would have learned in grade 1 mathematics. It's equivalent to Python's ==.

means "equality". It is NOT an assignment operator. It is a boolean operator. It is the = you would have learned in grade 1 mathematics. It's equivalent to Python's ==.

means "defined to be". It is written as == in ASCII. You use it to created named definitions of things.

means "equality". It is NOT an assignment operator. It is a boolean operator. It is the = you would have learned in grade 1 mathematics. It's equivalent to Python's ==.

 \triangleq means "defined to be". It is written as == in ASCII. You use it to created named definitions of things.

$$MyDef == A / \ B$$

At any time, **MyDef** will be the value of **A** and ed with **B**

It is a little like using a macro, or an inline.

```
TRUE \/ (TRUE /\ FALSE)

TRUE /\ (FALSE \/ (TRUE \/ FALSE))

FALSE /\ (TRUE /\ ((FALSE \/ TRUE)\/(TRUE \/ FALSE)))
```

TRUE

TRUE \/ (TRUE /\ FALSE)

TRUE /\ (FALSE \/ (TRUE \/ FALSE))

FALSE /\ (TRUE /\ ((FALSE \/ TRUE)\/(TRUE \/ FALSE)))

TRUE \/ (TRUE /\ FALSE)

TRUE /\ (FALSE \/ (TRUE \/ FALSE))

TRUE

TRUE /\ (FALSE \/ TRUE)\/(TRUE \/ FALSE)))

TRUE \/ (TRUE /\ FALSE)

TRUE

TRUE /\ (FALSE \/ (TRUE \/ FALSE))

TRUE

FALSE /\ (TRUE /\ ((FALSE \/ TRUE)\/(TRUE \/ FALSE)))

FALSE

TLA+ specs make heavy use of formulas like this

A /\ (B /\ ((C \/ D)\/(E /\ F))) /\ (G \/ H \/ I)

TLA+ specs make heavy use of formulas like this

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

TLA+ specs make heavy use of formulas like this

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

TLA+ specs make heavy use of formulas like this

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

TLA+ specs make heavy use of formulas like this

A /\ (B /\ ((C \/ D)\/(E /\ F))) /\ (G \/ H \/ I)

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

TLA+ specs make heavy use of formulas like this

A /\ (B /\ ((C \/ D)\/(E /\ F))) /\ (G \/ H \/ I)

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

TLA+ specs make heavy use of formulas like this

A /\ (B /\ ((C \/ D)\/(E /\ F))) /\ (G \/ H \/ I)

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

TLA+ specs make heavy use of formulas like this

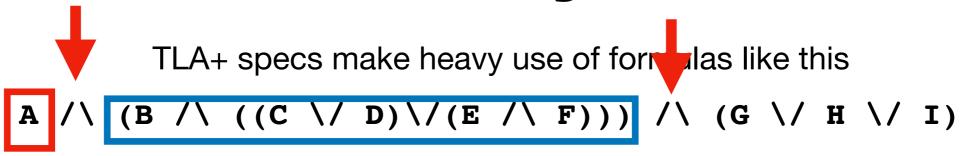
A /\ (B /\ ((C \/ D)\/(E /\ F))) /\ (G \/ H \/ I)

This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

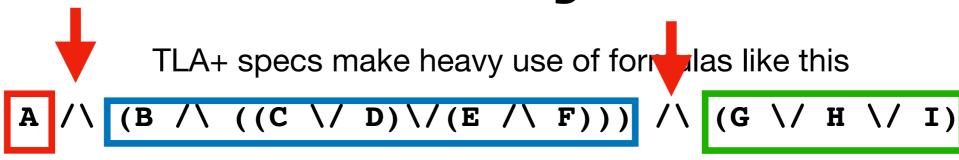
TLA+ specs make heavy use of formulas like this

A /\ (B /\ ((C \/ D)\/(E /\ F))) /\ (G \/ H \/ I)

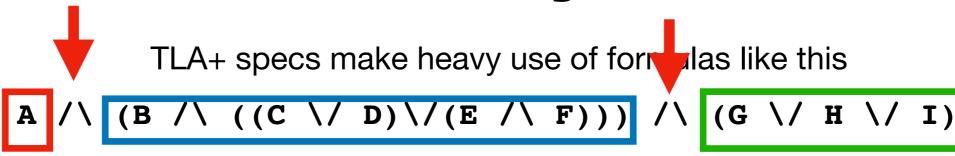
This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.



This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.



This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.



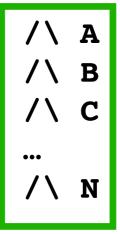
This sucks. TLA requires lots of Boolean logic, and these brackets are not fun. TLA+ has a better way to do it.

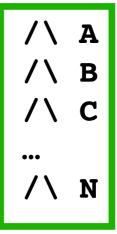
In general:

\/ A

In general:

\/ A







TLA+ does not allow "ambiguous" formulas:

A /\ B \/ C /\ D

TLA+ does not allow "ambiguous" formulas:

A /\ B \/ C /\ D

Mathematically, there is a correct answer, but people have a hard time remembering.

TLA+ forces you to be explicit.

TLA+ does not allow "ambiguous" formulas:

Mathematically, there is a correct answer, but people have a hard time remembering.

TLA+ forces you to be explicit.

$$(A /\ B) /\ (C /\ D)$$

TLA+ does not allow "ambiguous" formulas:

Mathematically, there is a correct answer, but people have a hard time remembering.

TLA+ forces you to be explicit.

$$(A /\ B) /\ (C /\ D)$$

(We'll come back to this.)

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

- The indentation level is meaningful.
- Expressions on the same indent level are "and"ed and "or"ed together.

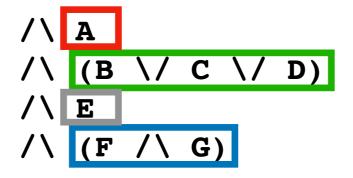
- This formula has four items at the top level, being AND'ed together. Let's rewrite using TLA+ syntax.
- We'll put \ at the outermost indent level.

A /\ (B \/ C \/ D) /\ E /\ (F /\ G)



/\ A

```
/\ A
/\ (B \/ C \/ D)
/\ E
/\ (F /\ G)
```



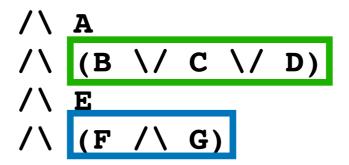
Now we can easily see all the "outermost" or "top-level" items being AND'ed together

We have more parentheses.

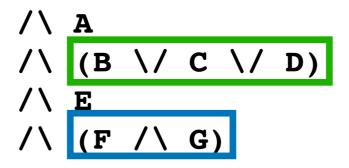
Now we can easily see all the "outermost" or "top-level" items being AND'ed together

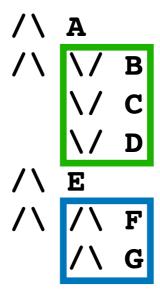
We have more parentheses.

Let's get rid of those too

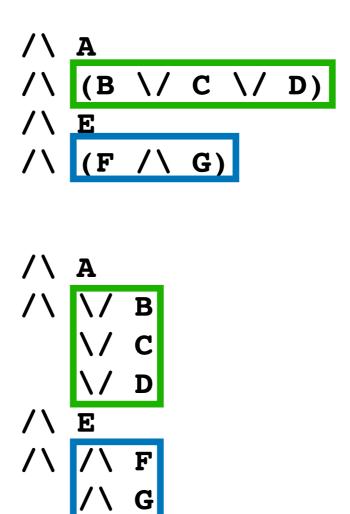


```
/\ A
/\ (B \/ C \/ D)
/\ E
/\ (F /\ G)
```





We'll create indent levels for each of these



The structure of the formula should be much more clear now. It creates an explicit graph-like visual structure out of parentheses.

Or think of it like bullet-points, if that helps.

What about this? How would this look?

A /\ B \/ C /\ D

What about this? How would this look?

A /\ B \/ C /\ D

TLA+ does not allow this. Precedence between \lor and \land must be made explicit with parentheses.

What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.

$$(A /\ B) /\ (C /\ D)$$

What about this? How would this look?

A /\ B \/ C /\ D

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.

 $(A /\ B) /\ (C /\ D)$

What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.

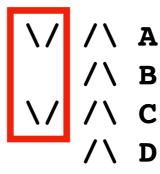
$$(A /\ B) /\ (C /\ D)$$

What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.

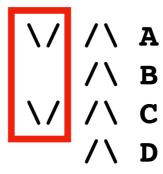
What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.



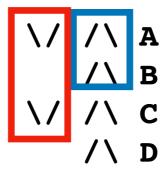
What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.



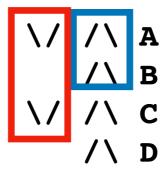
What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.



What about this? How would this look?

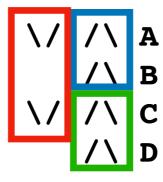
TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.



What about this? How would this look?

TLA+ does not allow this. Precedence between ∨ and ∧ must be made explicit with parentheses.





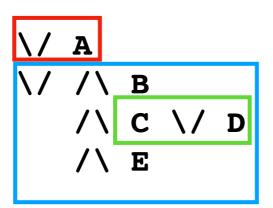
Exercise

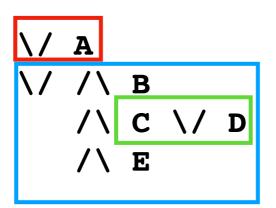
1. A $\ \ (B \ \ (C \ \ D) \ \ \ E)$

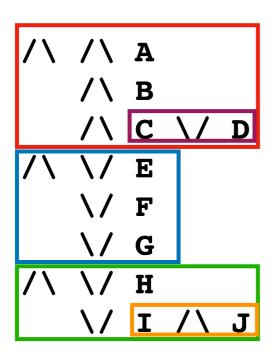
2. (A /\ B /\ (C \/ D)) /\ (E \/ F \/ G) /\ (H \/ (I /\ J))

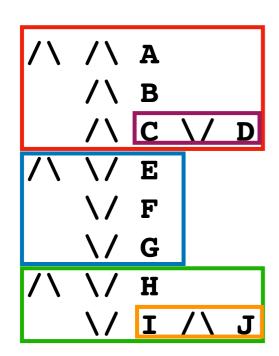
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))

```
1. A \/ (B /\ (C \/ D) /\ E)
```









```
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))

/\ \/ A
   \/ B /\ C
   \/ D

/\ E

/\ F

/\ \ G
   \/ H
   \/ I /\ J
```

```
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))
```

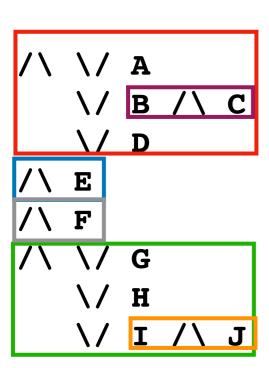
```
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))
```

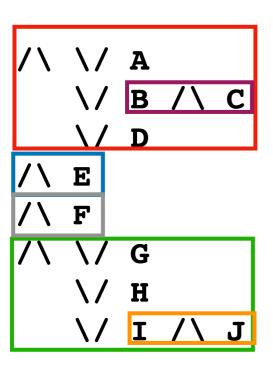
```
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))
```

```
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))
```

```
3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))
```

3. (A \/ (B /\ C) \/ D) /\ E /\ F /\ (G \/ H \/ (I /\ J))





• An unordered collection of unique things.

- An unordered collection of unique things.
- No item is ever duplicated in a set.

- An unordered collection of unique things.
- No item is ever duplicated in a set.
- {} is the "empty set".

- An unordered collection of unique things.
- No item is ever duplicated in a set.
- {} is the "empty set".
- {"a", "b", 123, {"a"}} is the set containing "a", "b", the number 123, and the set {"a"}.

- An unordered collection of unique things.
- No item is ever duplicated in a set.
- {} is the "empty set".
- {"a", "b", 123, {"a"}} is the set containing "a", "b", the number 123, and the set {"a"}.
- {"a", "b"} = {"b", "a"} (i.e. order does not matter).

- An unordered collection of unique things.
- No item is ever duplicated in a set.
- {} is the "empty set".
- {"a", "b", 123, {"a"}} is the set containing "a", "b", the number 123, and the set {"a"}.
- {"a", "b"} = {"b", "a"} (i.e. order does not matter).
- The concept maps to Python's set().

• ∈ is the "in set" operator, written as \in in ASCII

- ∈ is the "in set" operator, written as \in in ASCII
 - "a" ∈ {"a", "b"} is TRUE

- ∈ is the "in set" operator, written as \in in ASCII
 - "a" \in {"a", "b"} is TRUE
 - Python: "a" in set(["a", "b"])

- ∈ is the "in set" operator, written as \in in ASCII
 - "a" \in {"a", "b"} is TRUE
 - Python: "a" in set(["a", "b"])
 - "a" ∈ {"b", "c"} is FALSE

- ∈ is the "in set" operator, written as \in in ASCII
 - "a" \in {"a", "b"} is TRUE
 - Python: "a" in set(["a", "b"])
 - "a" ∈ {"b", "c"} is FALSE
 - Python: "a" in set(["b", "c"])

• ⊆ is the "subset" operator, written as \subseteq in ASCII

- ⊆ is the "subset" operator, written as \subseteq in ASCII
 - {"a"} ⊆ {"a", "b", "c"} is TRUE

- ⊆ is the "subset" operator, written as \subseteq in ASCII
 - {"a"} ⊆ {"a", "b", "c"} is TRUE
 - Python:
 set(["a"]).issubset(set(["a","b","c"]))

- ⊆ is the "subset" operator, written as \subseteq in ASCII
 - {"a"} ⊆ {"a", "b", "c"} is TRUE
 - Python: set(["a"]).issubset(set(["a","b","c"]))
 - {"a", "b"} ⊆ {"b", "c"} is FALSE

- ⊆ is the "subset" operator, written as \subseteq in ASCII
 - {"a"} ⊆ {"a", "b", "c"} is TRUE
 - Python: set(["a"]).issubset(set(["a","b","c"]))
 - {"a", "b"} ⊆ {"b", "c"} is FALSE
 - Python: set(["a","b"]).issubset(set(["b","c"]))

• ∪ is the "union" operator, written as \union in ASCII

- ∪ is the "union" operator, written as \union in ASCII
- It "squishes together" two sets to create a new set

- ∪ is the "union" operator, written as \union in ASCII
- It "squishes together" two sets to create a new set
 - $\{\text{"a", "b"}\} \cup \{\} = \{\text{"a", "b"}\}$

- ∪ is the "union" operator, written as \union in ASCII
- It "squishes together" two sets to create a new set
 - $\{\text{"a", "b"}\} \cup \{\} = \{\text{"a", "b"}\}$
 - $\{\text{"a", "b"}\} \cup \{\text{"c", "d"}\} = \{\text{"a", "b", "c", "d"}\}$

- ∪ is the "union" operator, written as \union in ASCII
- It "squishes together" two sets to create a new set
 - $\{\text{"a", "b"}\} \cup \{\} = \{\text{"a", "b"}\}$
 - $\{\text{"a", "b"}\} \cup \{\text{"c", "d"}\} = \{\text{"a", "b", "c", "d"}\}$
 - $\{\text{"a", "b"}\} \cup \{\text{"b", "c"}\} = \{\text{"a", "b", "c"}\}$

- ∪ is the "union" operator, written as \union in ASCII
- It "squishes together" two sets to create a new set
 - $\{"a", "b"\} \cup \{\} = \{"a", "b"\}$
 - $\{\text{"a", "b"}\} \cup \{\text{"c", "d"}\} = \{\text{"a", "b", "c", "d"}\}$
 - $\{"a", "b"\} \cup \{"b", "c"\} = \{"a", "b", "c"\}$
 - Python:
 set(["a","b"]).union(set(["b", "c"]))

• ∩ is the "intersect" operator, written as \intersect in ASCII

- ∩ is the "intersect" operator, written as \intersect in ASCII
- It returns a new set containing the elements common to both sets

- ∩ is the "intersect" operator, written as \intersect in ASCII
- It returns a new set containing the elements common to both sets
 - $\{\text{"a", "b"}\} \cap \{\} = \{\}$

- ∩ is the "intersect" operator, written as \intersect in ASCII
- It returns a new set containing the elements common to both sets
 - $\{\text{"a", "b"}\} \cap \{\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"c", "d"}\} = \{\}$

- ∩ is the "intersect" operator, written as \intersect in ASCII
- It returns a new set containing the elements common to both sets
 - $\{\text{``a''}, \text{``b''}\} \cap \{\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"c", "d"}\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"b", "c"}\} = \{\text{"b"}\}$

- ∩ is the "intersect" operator, written as \intersect in ASCII
- It returns a new set containing the elements common to both sets
 - $\{\text{``a''}, \text{``b''}\} \cap \{\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"c", "d"}\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"b", "c"}\} = \{\text{"b"}\}$
 - Python:

- n is the "intersect" operator, written as \intersect in ASCII
- It returns a new set containing the elements common to both sets
 - $\{\text{"a", "b"}\} \cap \{\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"c", "d"}\} = \{\}$
 - $\{\text{"a", "b"}\} \cap \{\text{"b", "c"}\} = \{\text{"b"}\}$
 - Python:
 - set(["a", "b"]).intersection(set(["b", "c"]))

• .. is roughly equivalent to Python's range() function

- .. is roughly equivalent to Python's range() function
 - $1..10 = \{1,2,3,4,5,6,7,8,9,10\}$

- .. is roughly equivalent to Python's range() function
 - $1..10 = \{1,2,3,4,5,6,7,8,9,10\}$
 - In Python: set(range(1,11))

(TRUE or FALSE for each)

$$\{1,2,3\} \subseteq (\{1,2,3\} \cap \{2,3\})$$

"a"
$$\in$$
 (({"b", "c", "d"} \cap {"e", "f"}) \cup {"a"})

$$("a" \in {"a", "b"}) \lor ("b" \in {"c", "d"})$$

(TRUE or FALSE for each)

TRUE

$$\{1,2,3\} \subseteq (\{1,2,3\} \cap \{2,3\})$$

"a"
$$\in$$
 (({"b", "c", "d"} \cap {"e", "f"}) \cup {"a"})

$$("a" \in {"a", "b"}) \lor ("b" \in {"c", "d"})$$

(TRUE or FALSE for each)

TRUE

$$\{1,2,3\} \subseteq (\{1,2,3\} \cap \{2,3\})$$

FALSE

"a"
$$\in$$
 (({"b", "c", "d"} \cap {"e", "f"}) \cup {"a"})

$$("a" \in {"a", "b"}) \lor ("b" \in {"c", "d"})$$

(TRUE or FALSE for each)

$$\{1,2,3\} \subseteq (\{1,2,3\} \cap \{2,3\})$$

"a"
$$\in$$
 (({"b", "c", "d"} \cap {"e", "f"}) \cup {"a"})

$$("a" \in {"a", "b"}) \lor ("b" \in {"c", "d"})$$

TRUE

FALSE

TRUE

(TRUE or FALSE for each)

$$\{\text{"a", "b"}\} \subseteq \{\text{"b", "a", "c"}\}\$$

$$\{1,2,3\} \subseteq (\{1,2,3\} \cap \{2,3\})$$
 FALSE

"a"
$$\in ((\{"b", "c", "d"\} \cap \{"e", "f"\}) \cup \{"a"\})$$

$$("a" \in \{"a", "b"\}) \lor ("b" \in \{"c", "d"\})$$

• ∃ means "there exists", written as **E** in ASCII

- ∃ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$

- means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means "there exists some x in the set S such that P(x) is TRUE"

- ∃ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"

- ∃ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"

- ■ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"

- ■ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"

- ∃ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"
 - The entire expression evaluates to TRUE or FALSE

- ∃ means "there exists", written as \E in ASCII
- The usual form is
- $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"
 - The entire expression evaluates to TRUE or FALSE
 - $\exists x \in \{1,2,3,4\} : x > 3$ is TRUE

- ∃ means "there exists", written as \E in ASCII
- The usual form is $\exists x \in S : P(x)$
 - This means 'there exists some x in the set S such that P(x) is TRUE"
 - The entire expression evaluates to TRUE or FALSE
 - $\exists x \in \{1,2,3,4\} : x > 3$ is TRUE
 - $\exists x \in \{1,2,3,4\} : x > 5$ is FALSE

$$\exists x \in \{1,2,3,4,5\} : x > 5$$

```
\exists x \in \{1,2,3,4,5\} : x > 5
```

```
def exists(S):
    for x in S:
        if x > 5:
        return True
    return False
```

```
\exists x \in \{1,2,3,4,5\} : x > 5
```

```
def exists(S):
    for x in S:
        if x > 5:
            return True
    return False

exists(set([1,2,3,4,5])) # False
```

```
\exists x \in \{1,2,3,4,5\} : x > 5
```

is roughly equivalent to the following Python expressions

def exists(S):

```
for x in S:
    if x > 5:
        return True
    return False

exists(set([1,2,3,4,5])) # False

any(map(lambda x: x > 5, [1,2,3,4,5])) #False
```

 • means "for all", written as \A in ASCII

- ∀ means "for all", written as \A in ASCII
- The usual form is $\forall x \in S : P(x)$

- • means "for all", written as \A in ASCII
- The usual form is $\forall x \in S : P(x)$
 - This means "for all x in the set S, it is the case that P(x) is TRUE"

- ∀ means "for all", written as \A in ASCII
- The usual form is $\forall x \in S : P(x)$
 - This means 'for all x in the set S, it is the case that P(x) is TRUE"

- ▼ means "for all", written as \A in ASCII
- The usual form is $\forall x \in S : P(x)$
 - This means 'for all x in the set S, it is the case that P(x) is TRUE"

- ▼ means "for all", written as \A in ASCII
- The usual form is $\forall x \in S : P(x)$
 - This means 'for all x in the set S, it is the case that P(x) is TRUE"

- • means "for all", written as \A in ASCII
- The usual form is $\forall x \in S$: P(x)
 - This means 'for all x in the set S, it is the case that P(x) is TRUE"

- ▼ means "for all", written as \A in ASCII
- The usual form is $\forall x \in S : P(x)$
 - This means 'for all x in the set S, it is the case that P(x) is TRUE"
 - The entire expression evaluates to TRUE or FALSE

- ▼ means "for all", written as \A in ASCII
- The usual form is

$$\forall x \in S : P(x)$$

- This means 'for all x in the set S, it is the case that P(x) is TRUE"
- The entire expression evaluates to TRUE or FALSE
- $\forall x \in \{1,2,3,4\} : x > 3$ is FALSE

- ▼ means "for all", written as \A in ASCII
- The usual form is
- $\forall x \in S : P(x)$
 - This means 'for all x in the set S, it is the case that P(x) is TRUE"
 - The entire expression evaluates to TRUE or FALSE
 - $\forall x \in \{1,2,3,4\} : x > 3$ is FALSE
 - $\forall x \in \{1,2,3,4\} : x > 0$ is TRUE

 $\forall x \in \{1,2,3,4,5\} : x > 5$

 $\forall x \in \{1,2,3,4,5\} : x > 5$

```
def for_all(S):
    for x in S:
        if not (x > 5):
           return False
    return True
```

```
\forall x \in \{1,2,3,4,5\} : x > 5
```

```
def for_all(S):
    for x in S:
        if not (x > 5):
           return False
    return True

for all(set([1,2,3,4,5])) # False
```

```
\forall x \in \{1,2,3,4,5\} : x > 5
```

```
def for_all(S):
    for x in S:
        if not (x > 5):
           return False
    return True

for_all(set([1,2,3,4,5])) # False
```

```
all(map(lambda x: x > 5, [1,2,3,4,5])) # False
```

$$\exists x \in \{1,2,3,4\} : (x > 3) \land (x < 4)$$

$$\exists x \in \{1,2,3,4\} : \exists y \in \{5,6,7\} : x < y$$

$$\forall x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\exists x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\forall x \in \{3,4,5\} : \land x > 1$$

$$\land x < 6$$

$$\exists x \in \{1,2,3,4\} : (x > 3) \land (x < 4)$$

FALSE

$$\exists x \in \{1,2,3,4\} : \exists y \in \{5,6,7\} : x < y$$

$$\forall x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\exists x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\forall x \in \{3, 4, 5\} : \land x > 1$$
 $\land x < 6$

$$\exists x \in \{1,2,3,4\} : (x > 3) \land (x < 4)$$

$$\exists x \in \{1,2,3,4\} : \exists y \in \{5,6,7\} : x < y$$
 TRUE

FALSE

$$\forall x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\exists x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\forall x \in \{3, 4, 5\} : \land x > 1$$
 $\land x < 6$

$$\exists x \in \{1,2,3,4\} : (x > 3) \land (x < 4)$$
 FALSE

$$\exists x \in \{1,2,3,4\} : \exists y \in \{5,6,7\} : x < y$$
 TRUE

$$\forall x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$
 FALSE

$$\exists x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$

$$\forall x \in \{3, 4, 5\} : \land x > 1$$
 $\land x < 6$

$$\exists x \in \{1,2,3,4\} : (x > 3) \land (x < 4)$$
 FALSE

$$\exists x \in \{1,2,3,4\} : \exists y \in \{5,6,7\} : x < y$$
 TRUE

$$\forall x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$
 FALSE

$$\exists x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$
 TRUE

$$\forall x \in \{3, 4, 5\} : \land x > 1$$
 $\land x < 6$

$$\exists x \in \{1,2,3,4\} : (x > 3) \land (x < 4)$$
 FALSE

$$\exists x \in \{1,2,3,4\} : \exists y \in \{5,6,7\} : x < y$$
 TRUE

$$\forall x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$
 FALSE

$$\exists x \in \{1,2,3\} : \{x\} \subseteq \{1,2\}$$
 TRUE

$$\forall x \in \{3, 4, 5\} : \land x > 1$$

 $\wedge x < 6$

Set constructors

Set constructors

 $\{x \in S : P\}$ is like a **filter()** function, creating a new set consisting of the elements of **S** that satisfy **P**.

 $\{x \in S : P\}$ is like a **filter()** function, creating a new set consisting of the elements of **S** that satisfy **P**.

$$\{ x \in \{1,2,3,4,5\} : x > 3 \} = \{4,5\}$$

 $\{x \in S : P\}$ is like a **filter()** function, creating a new set consisting of the elements of **S** that satisfy **P**.

$$\{ x \in \{1,2,3,4,5\} : x > 3 \} = \{4,5\}$$

This is equivalent to the following Python expressions:

 $\{x \in S : P\}$ is like a **filter()** function, creating a new set consisting of the elements of **S** that satisfy **P**.

$$\{ x \in \{1,2,3,4,5\} : x > 3 \} = \{4,5\}$$

This is equivalent to the following Python expressions:

$$set([x for x in [1,2,3,4,5] if x > 3])$$

 $\{x \in S : P\}$ is like a **filter()** function, creating a new set consisting of the elements of **S** that satisfy **P**.

$$\{ x \in \{1,2,3,4,5\} : x > 3 \} = \{4,5\}$$

This is equivalent to the following Python expressions:

$$set([x for x in [1,2,3,4,5] if x > 3])$$

```
set(filter(lambda x: x > 3, [1,2,3,4,5]))
```

 $\{e: x \in S\}$ is like a map() function, applying e to every element of S.

$$\{x * 2 : x \in \{1,2,3,4,5\}\} = \{2,4,6,8,10\}$$

 $\{e: x \in S\}$ is like a map() function, applying e to every element of S.

$$\{x * 2 : x \in \{1,2,3,4,5\}\} = \{2,4,6,8,10\}$$

This is roughly equivalent to the following Python expressions:

 $\{e: x \in S\}$ is like a map() function, applying e to every element of S.

$$\{x * 2 : x \in \{1,2,3,4,5\}\} = \{2,4,6,8,10\}$$

This is roughly equivalent to the following Python expressions:

 $\{e: x \in S\}$ is like a map() function, applying e to every element of S.

$$\{x * 2 : x \in \{1,2,3,4,5\}\} = \{2,4,6,8,10\}$$

This is roughly equivalent to the following Python expressions:

```
set([x*2 for x in [1,2,3,4,5]])
```

```
set(map(lambda x: x*2, [1,2,3,4,5]))
```

Exercises

- Write a set constructor that creates a set of elements from {1,2,3,4,5}, consisting of the elements that are greater than 1 and less than 5 (i.e. we want {2, 3, 4}).
- 2. Write a set constructor that creates a set of elements from **{1,2,3,4,5}**, consisting of the elements that, when multiplied by **3**, are greater than **10** (i.e. we want **{4, 5}**).
- 3. Write a set constructor that creates a set of elements consisting of each element of **{1, 2, 3, 4, 5}** added to itself and then subtracting **1** (i.e. we want **{1, 3, 5, 7, 9}**).

```
1. \{x \in \{1,2,3,4,5\} : \land x > 1  or \{x \in \{1,2,3,4,5\} : x > 1 \land x < 5\}
```

1.
$$\{x \in \{1,2,3,4,5\} : \land x > 1$$
 or $\{x \in \{1,2,3,4,5\} : x > 1 \land x < 5\}$

2.
$$\{x \in \{1,2,3,4,5\} : x*3 > 10\}$$

1.
$$\{x \in \{1,2,3,4,5\} : \land x > 1$$
 or $\{x \in \{1,2,3,4,5\} : x > 1 \land x < 5\}$

2.
$$\{x \in \{1,2,3,4,5\} : x*3 > 10\}$$

3.
$$\{x+x-1: x \in \{1,2,3,4,5\}\}$$