Math Time!!!

“and” / “or”

“and” / “or”

$\wedge$ 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”

$\wedge$ 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
```

$\vee$ 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
FALSE \/ TRUE  = TRUE
```

$=$ and $\triangleq$

= and ≜

=	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 `==`.

= and $\triangleq$

=	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.

= and ≜

= 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.

```
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.

Exercise

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

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

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

Exercise

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

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

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

Exercise

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

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

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

Exercise

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

TLA + Syntax

TLA+ specs make heavy use of formulas like this

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

TLA + Syntax

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 + Syntax

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.

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

TLA + Syntax

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.

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

Indentation level determines which variables are AND'ed and OR'ed together
It's like using a bulleted-list of $\wedge$ or $\vee$ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND'ed and OR'ed together
It's like using a bulleted-list of $\land$ or $\lor$ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND’ed and OR’ed together
It’s like using a bulleted-list of /\ or \/ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND’ed and OR’ed together
It’s like using a bulleted-list of /\ or \/ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND'ed and OR'ed together
It's like using a bulleted-list of /\ or \/ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND’ed and OR’ed together
It’s like using a bulleted-list of /\ or \/ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND'ed and OR'ed together
It's like using a bulleted-list of $\wedge$ or $\vee$ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND'ed and OR'ed together
It's like using a bulleted-list of $\land$ or $\lor$ symbols

TLA + Syntax

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.

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

Indentation level determines which variables are AND'ed and OR'ed together
It's like using a bulleted-list of $\wedge$ or $\vee$ symbols

TLA+ Syntax

In general:

TLA+ Syntax

In general:

```
A /\ B /\ C /\ … /\ N        =        /\ A
                                      /\ B
                                      /\ C
                                      …
                                      /\ N
```

TLA+ Syntax

In general:

```
A /\ B /\ C /\ … /\ N        =        /\ A
                                      /\ B
                                      /\ C
                                      …
                                      /\ N
```

```
A \/ B \/ C \/ … \/ N        =        \/ A
                                      \/ B
                                      \/ C
                                      …
                                      \/ N
```

TLA+ Syntax

In general:

$$\boxed{\texttt{A /\textbackslash{} B /\textbackslash{} C /\textbackslash{} … /\textbackslash{} N}}$$

```
A /\ B /\ C /\ … /\ N      =      /\ A
                                  /\ B
                                  /\ C
                                  …
                                  /\ N
```

```
A \/ B \/ C \/ … \/ N      =      \/ A
                                  \/ B
                                  \/ C
                                  …
                                  \/ N
```

TLA+ Syntax

In general:

$\boxed{\texttt{A /\textbackslash{} B /\textbackslash{} C /\textbackslash{} ... /\textbackslash{} N}}$ =

```
/\ A
/\ B
/\ C
…
/\ N
```

```
A \/ B \/ C \/ … \/ N
```

=

```
\/ A
\/ B
\/ C
…
\/ N
```

TLA+ Syntax

In general:

```
A /\ B /\ C /\ … /\ N
```

=

```
/\ A
/\ B
/\ C
…
/\ N
```

```
A \/ B \/ C \/ … \/ N
```

=

```
\/ A
\/ B
\/ C
…
\/ N
```

TLA+ Syntax

In general:

```
A /\ B /\ C /\ ... /\ N
```

=

```
/\ A
/\ B
/\ C
...
/\ N
```

```
A \/ B \/ C \/ ... \/ N
```

=

```
\/ A
\/ B
\/ C
...
\/ N
```

TLA+ Syntax

TLA+ does not allow “ambiguous” formulas:

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

TLA+ Syntax

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+ Syntax

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.

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

TLA+ Syntax

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.

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

(We’ll come back to this.)

TLA+ Syntax

- The indentation level is meaningful.

- Expressions on the same indent level are “and”ed and “or”ed together.

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

TLA+ Syntax

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

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

- This formula has four items at the top level, being AND’ed together. Let’s rewrite using TLA+ syntax.

TLA+ Syntax

- The indentation level is meaningful.

- Expressions on the same indent level are “and”ed and “or”ed together.

$$\boxed{\texttt{A}}\ \texttt{/\textbackslash\ (B \textbackslash/ C \textbackslash/ D) /\textbackslash\ E /\textbackslash\ (F /\textbackslash\ G)}$$

- This formula has four items at the top level, being AND’ed together. Let’s rewrite using TLA+ syntax.

TLA+ Syntax

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

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

- This formula has four items at the top level, being AND’ed together. Let’s rewrite using TLA+ syntax.

TLA+ Syntax

- The indentation level is meaningful.

- Expressions on the same indent level are “and”ed and “or”ed together.

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

- This formula has four items at the top level, being AND’ed together. Let’s rewrite using TLA+ syntax.

TLA+ Syntax

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

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

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

TLA+ Syntax

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

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

- 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.

TLA+ Syntax

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

TLA+ Syntax

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

```
/\ A
```

TLA+ Syntax

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

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

TLA+ Syntax

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

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

TLA+ Syntax

$\boxed{\texttt{A}}$ `/\` $\boxed{\texttt{(B \textbackslash/ C \textbackslash/ D)}}$ `/\` $\boxed{\texttt{E}}$ `/\` $\boxed{\texttt{(F /\textbackslash\ G)}}$

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

TLA+ Syntax

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

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

Now we can easily see all the “outermost” or “top-level” items being AND’ed together

TLA+ Syntax

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

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

Now we can easily see all the “outermost” or “top-level” items being AND’ed together

TLA+ Syntax

$\boxed{\texttt{A}}$ `/\` $\boxed{\texttt{(B \textbackslash/ C \textbackslash/ D)}}$ `/\` $\boxed{\texttt{E}}$ `/\` $\boxed{\texttt{(F /\textbackslash\ G)}}$

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

Now we can easily see all the “outermost” or “top-level” items being AND’ed together

TLA+ Syntax

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

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

Now we can easily see all the “outermost” or “top-level” items being AND’ed together

TLA+ Syntax

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

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

Now we can easily see all the “outermost” or “top-level” items being AND’ed together

TLA+ Syntax

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

```
/\ 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.

TLA+ Syntax

$\boxed{\texttt{A}}$ `/\` $\boxed{\texttt{(B \textbackslash/ C \textbackslash/ D)}}$ `/\` $\boxed{\texttt{E}}$ `/\` $\boxed{\texttt{(F /\textbackslash\ G)}}$

```
/\ 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.

Let's get rid of those too

TLA+ Syntax

We'll create indent levels for each of these

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

TLA+ Syntax

We'll create indent levels for each of these

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


/\ A
```

TLA+ Syntax

We'll create indent levels for each of these

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

```
/\ A
/\ \/ B
```

TLA+ Syntax

We'll create indent levels for each of these

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

```
/\ A
/\ \/ B
   \/ C
```

TLA+ Syntax

We'll create indent levels for each of these

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

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

TLA+ Syntax

We'll create indent levels for each of these

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

```
/\ A
/\ \/ B
   \/ C
   \/ D
/\ E
```

TLA+ Syntax

We'll create indent levels for each of these

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

```
/\ A
/\ \/ B
   \/ C
   \/ D
/\ E
/\ /\ F
```

TLA+ Syntax

We'll create indent levels for each of these

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

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

TLA+ Syntax

We'll create indent levels for each of these

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

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

TLA+ Syntax

We'll create indent levels for each of these

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

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

TLA+ Syntax

We'll create indent levels for each of these

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

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

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 .

TLA+ Syntax

What about this? How would this look?

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

TLA+ Syntax

What about this? How would this look?

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

TLA+ does not allow this. Precedence between $\vee$ and $\wedge$ must be made explicit with parentheses.

TLA+ Syntax

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)
```

TLA+ Syntax

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)
```

Which becomes

TLA+ Syntax

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)
```

Which becomes

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

TLA+ Syntax

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)
```

Which becomes

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

TLA+ Syntax

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)
```

Which becomes

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

TLA+ Syntax

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)
```

Which becomes

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

TLA+ Syntax

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)
```

Which becomes

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

TLA+ Syntax

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.

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

Which becomes

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

TLA+ Syntax

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)
```

Which becomes

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

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))`

Solutions

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

```
\/ A
\/ /\ B
   /\ C \/ D
   /\ E
```

Solutions

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

```
\/ A
\/ /\ B
   /\ C \/ D
   /\ E
```

Solutions

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

```
\/ A
\/ /\ B
   /\ C \/ D
   /\ E
```

Solutions

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

```
\/ A
\/ /\ B
   /\ C \/ D
   /\ E
```

Solutions

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

```
\/ A
\/ /\ B
   /\ C \/ D
   /\ E
```

Alternate Solution

```
\/ A
\/ /\ B
   /\ \/ C
      \/ D
   /\ E
```

Solutions

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

```
/\ /\ A
   /\ B
   /\ C \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ I /\ J
```

Solutions

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

```
/\  /\  A
    /\  B
    /\  C  \/  D
/\  \/  E
    \/  F
    \/  G
/\  \/  H
    \/  I  /\  J
```

Solutions

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

```
/\ /\ A
   /\ B
   /\ C \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ I /\ J
```

Solutions

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

```
/\ /\ A
   /\ B
   /\ C \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ I /\ J
```

Solutions

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

```
/\ /\ A
   /\ B
   /\ C \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ I /\ J
```

Solutions

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

```
/\ /\ A
   /\ B
   /\ C \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ I /\ J
```

Solutions

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

```
/\ /\ A
   /\ B
   /\ C \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ I /\ J
```

Alternate Solution

```
/\ /\ A
   /\ B
   /\ \/ C
      \/ D
/\ \/ E
   \/ F
   \/ G
/\ \/ H
   \/ /\ I
      /\ J
```

Solutions

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


/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

```
/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

```
/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Solutions

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

```
/\ \/ A
   \/ B /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ I /\ J
```

Alternate Solution

```
/\ \/ A
   \/ /\ B
      /\ C
   \/ D
/\ E
/\ F
/\ \/ G
   \/ H
   \/ /\ I
      /\ J
```

Sets

Sets

- An unordered collection of unique things.

Sets

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

Sets

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

Sets

- 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”}.

Sets

- 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).

Sets

- 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()`.

Sets

Sets

- $\in$ is the "in set" operator, written as **`\in`** in ASCII

Sets

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

Sets

- $\in$ is the “in set” operator, written as **`\in`** in ASCII
  - “a” $\in$ {“a”, “b”} is TRUE
    - Python: `“a” in set([“a”, “b”])`

Sets

- $\in$ is the “in set” operator, written as **`\in`** in ASCII
  - “a” $\in$ {“a”, “b”} is TRUE
    - Python: `“a” in set([“a”, “b”])`
  - “a” $\in$ {“b”, “c”} is FALSE

Sets

- $\in$ is the “in set” operator, written as **`\in`** in ASCII
  - “a” $\in$ {“a”, “b”} is TRUE
    - Python: `“a” in set([“a”, “b”])`
  - “a” $\in$ {“b”, “c”} is FALSE
    - Python: `“a” in set([“b”, “c”])`

Sets

Sets

- $\subseteq$ is the “subset” operator, written as **`\subseteq`** in ASCII

Sets

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

Sets

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

Sets

- ⊆ 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

Sets

- ⊆ 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”]))`

Sets

Sets

- $\cup$ is the "union" operator, written as **`\union`** in ASCII

Sets

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

Sets

- ∪ is the “union” operator, written as `\union` in ASCII
- It “squishes together” two sets to create a new set
  - {“a”, “b”} ∪ {} = {“a”, “b”}

Sets

- ∪ is the “union” operator, written as **`\union`** in ASCII
- It “squishes together” two sets to create a new set
  - {“a”, “b”} ∪ {} = {“a”, “b”}
  - {“a”, “b”} ∪ {“c”, “d”} = {“a”, “b”, “c”, “d”}

Sets

- ∪ is the “union” operator, written as **`\union`** in ASCII
- It “squishes together” two sets to create a new set
  - {“a”, “b”} ∪ {} = {“a”, “b”}
  - {“a”, “b”} ∪ {“c”, “d”} = {“a”, “b”, “c”, “d”}
  - {“a”, “b”} ∪ {“b”, “c”} = {“a”, “b”, “c”}

Sets

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

Sets

Sets

- $\cap$ is the "intersect" operator, written as **`\intersect`** in ASCII

Sets

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

Sets

- ∩ is the “intersect” operator, written as **`\intersect`** in ASCII
- It returns a new set containing the elements common to both sets
  - {“a”, “b”} ∩ {} = {}

Sets

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

Sets

- ∩ is the “intersect” operator, written as **`\intersect`** in ASCII
- It returns a new set containing the elements common to both sets
  - {“a”, “b”} ∩ {} = {}
  - {“a”, “b”} ∩ {“c”, “d”} = {}
  - {“a”, “b”} ∩ {“b”, “c”} = {“b”}

Sets

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

Sets

- ∩ is the “intersect” operator, written as **`\intersect`** in ASCII
- It returns a new set containing the elements common to both sets
  - {“a”, “b”} ∩ {} = {}
  - {“a”, “b”} ∩ {“c”, “d”} = {}
  - {“a”, “b”} ∩ {“b”, “c”} = {“b”}
  - Python:
    - `set([“a”, “b”]).intersection(set([“b”, “c”]))`

Sets

Sets

- `..` is roughly equivalent to Python's **`range()`** function

Sets

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

Sets

- `..` 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))`

Exercises

(TRUE or FALSE for each)

$\{\text{“a”}, \text{“b”}\} \subseteq \{\text{“b”}, \text{“a”}, \text{“c”}\}$

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

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

$(\text{“a”} \in \{\text{“a”}, \text{“b”}\}) \vee (\text{“b”} \in \{\text{“c”}, \text{“d”}\})$

Exercises

(TRUE or FALSE for each)

$\{\text{“a”}, \text{“b”}\} \subseteq \{\text{“b”}, \text{“a”}, \text{“c”}\}$ TRUE

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

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

$(\text{“a”} \in \{\text{“a”}, \text{“b”}\}) \vee (\text{“b”} \in \{\text{“c”}, \text{“d”}\})$

Exercises

(TRUE or FALSE for each)

{“a”, “b”} ⊆ {“b”, “a”, “c”} TRUE

{1,2,3} ⊆ ({1,2,3} ∩ {2,3}) FALSE

“a” ∈ (({“b”, “c”, “d”} ∩ {“e”, “f”}) ∪ {“a”})

(“a” ∈ {“a”, “b”}) ∨ (“b” ∈ {“c”, “d”})

Exercises

(TRUE or FALSE for each)

{“a”, “b”} ⊆ {“b”, “a”, “c”} TRUE

{1,2,3} ⊆ ({1,2,3} ∩ {2,3}) FALSE

“a” ∈ (({“b”, “c”, “d”} ∩ {“e”, “f”}) ∪ {“a”}) TRUE

(“a” ∈ {“a”, “b”}) ∨ (“b” ∈ {“c”, “d”})

Exercises

(TRUE or FALSE for each)

$\{\text{“a”}, \text{“b”}\} \subseteq \{\text{“b”}, \text{“a”}, \text{“c”}\}$	TRUE
$\{1,2,3\} \subseteq (\{1,2,3\} \cap \{2,3\})$	FALSE
$\text{“a”} \in ((\{\text{“b”}, \text{“c”}, \text{“d”}\} \cap \{\text{“e”}, \text{“f”}\}) \cup \{\text{“a”}\})$	TRUE
$(\text{“a”} \in \{\text{“a”}, \text{“b”}\}) \vee (\text{“b”} \in \{\text{“c”}, \text{“d”}\})$	TRUE

“There exists”

“There exists”

- $\exists$ means “there exists”, written as `\E` in ASCII

“There exists”

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

“There exists”

- $\exists$ 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”

“There exists”

- $\exists$ 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”

“There exists”

- $\exists$ means “there exists”, written as **`\E`** in ASCII
- The usual form is $\boxed{\exists}\ \boxed{x \in S} : P(x)$
  - This means “$\boxed{\text{there exists}}$ $\boxed{\text{some x in the set S}}$ such that P(x) is TRUE”

“There exists”

- $\exists$ means “there exists”, written as **`\E`** in ASCII
- The usual form is $\boxed{\exists}\ \boxed{x \in S}\ \boxed{:}\ P(x)$
  - This means “$\boxed{\text{there exists}}\ \boxed{\text{some x in the set S}}\ \boxed{\text{such that}}$ P(x) is TRUE”

“There exists”

- ∃ means “there exists”, written as **`\E`** in ASCII
- The usual form is $\boxed{\exists}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
    - This means “there exists some x in the set S such that P(x) is TRUE”

“There exists”

- $\exists$ means “there exists”, written as **`\E`** in ASCII
- The usual form is $\boxed{\exists}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
  - This means “there exists some x in the set S such that P(x) is TRUE”
  - The entire expression evaluates to TRUE or FALSE

“There exists”

- $\exists$ means “there exists”, written as **`\E`** in ASCII
- The usual form is $\boxed{\exists}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
  - 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

“There exists”

- ∃ means “there exists”, written as **`\E`** in ASCII
- The usual form is $\boxed{\exists}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
  - 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

“There exists”

“There exists”

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

is roughly equivalent to the following Python expressions

“There exists”

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

is roughly equivalent to the following Python expressions

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

“There exists”

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

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
```

“There exists”

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

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
```

“For all”

“For all”

- $\forall$ means “for all”, written as `\A` in ASCII

“For all”

- $\forall$ means “for all”, written as **`\A`** in ASCII
- The usual form is $\forall\ x \in S : P(x)$

“For all”

- $\forall$ 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”

“For all”

- $\forall$ means “for all”, written as **`\A`** in ASCII
- The usual form is $\boxed{\forall}\, x \in S : P(x)$
  - This means “for all x in the set S, it is the case that P(x) is TRUE”

“For all”

- $\forall$ means “for all”, written as **`\A`** in ASCII
- The usual form is $\boxed{\forall}\,\boxed{x \in S} : P(x)$
    - This means “$\boxed{\text{for all}}$ $\boxed{\text{x in the set S,}}$ it is the case that P(x) is TRUE”

"For all"

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

“For all”

- $\forall$ means “for all”, written as **`\A`** in ASCII
- The usual form is $\boxed{\forall}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
  - This means “for all x in the set S, it is the case that P(x) is TRUE”

“For all”

- $\forall$ means “for all”, written as **`\A`** in ASCII
- The usual form is $\boxed{\forall}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
  - 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

“For all”

- $\forall$ means “for all”, written as **`\A`** in ASCII
- The usual form is $\boxed{\forall}\ \boxed{x \in S}\ \boxed{:}\ \boxed{P(x)}$
  - 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

"For all"

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

“For all”

“For all”

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

is roughly equivalent to the following Python expressions

"For all"

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

is roughly equivalent to the following Python expressions

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

“For all”

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

is roughly equivalent to the following Python expressions

```
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
```

"For all"

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

is roughly equivalent to the following Python expressions

```
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
```

Exercises

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

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

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

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

$$\forall\ x \in \{3, 4, 5\} : \begin{array}{l} \wedge\ x > 1 \\ \wedge\ x < 6 \end{array}$$

Exercises

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

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

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

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

$$\forall\, x \in \{3, 4, 5\} : \begin{array}{l} \wedge\; x > 1 \\ \wedge\; x < 6 \end{array}$$

Exercises

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

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

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

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

$$\forall\, x \in \{3, 4, 5\} : \begin{array}{l} \wedge\ x > 1 \\ \wedge\ x < 6 \end{array}$$

Exercises

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

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

$$\forall\, x \in \{3, 4, 5\} : \begin{array}{l} \wedge\; x > 1 \\ \wedge\; x < 6 \end{array}$$

Exercises

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

$$\forall\, x \in \{3, 4, 5\} : \begin{array}{l} \wedge\ x > 1 \\ \wedge\ x < 6 \end{array}$$

Exercises

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

$$\forall\, x \in \{3, 4, 5\} : \begin{array}{l} \wedge\; x > 1 \\ \wedge\; x < 6 \end{array}$$ TRUE

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**.

Set constructors

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

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

Set constructors

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

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

This is equivalent to the following Python expressions:

Set constructors

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

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

This is equivalent to the following Python expressions:

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

Set constructors

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

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

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]))
```

Set constructors

Set constructors

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

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

Set constructors

$\{ e : x \in S \}$ 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 constructors

$\{ e : x \in S \}$ 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 constructors

{ e : x ∈ 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

1. 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}**).

Solutions

Solutions

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

Solutions

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

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

Solutions

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

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

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