

Binary Relations

Outline for Today

Binary Relations

- Reasoning about connections between objects.

Equivalence Relations

- Reasoning about clusters.

Strict Orders

- Reasoning about prerequisites.

Relationships

In CS103, you've seen examples of relationships

- between sets:

$$A \subseteq B$$

- between numbers:

$$x < y \quad x \equiv_k y \quad x \leq y$$

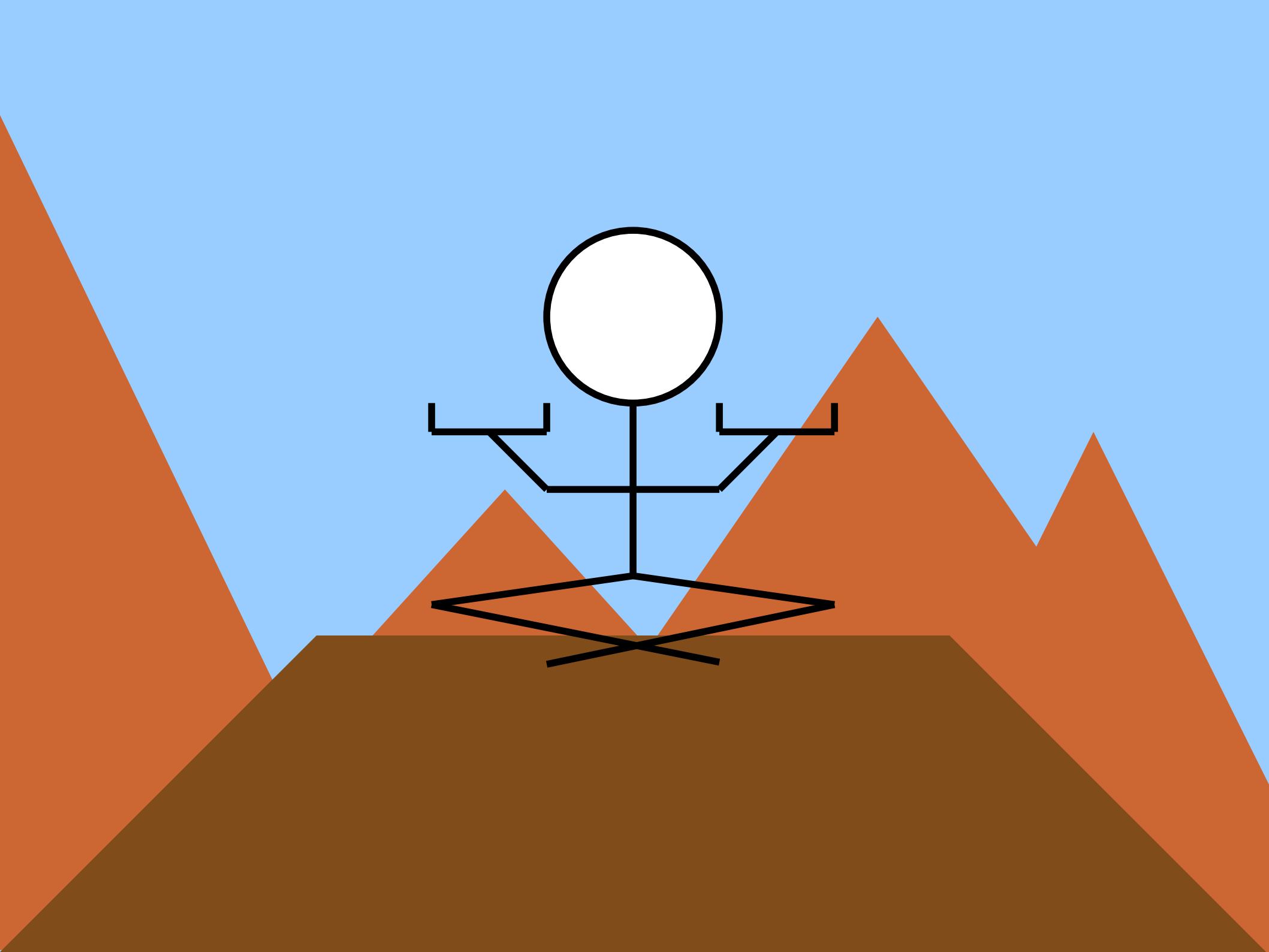
- between people:

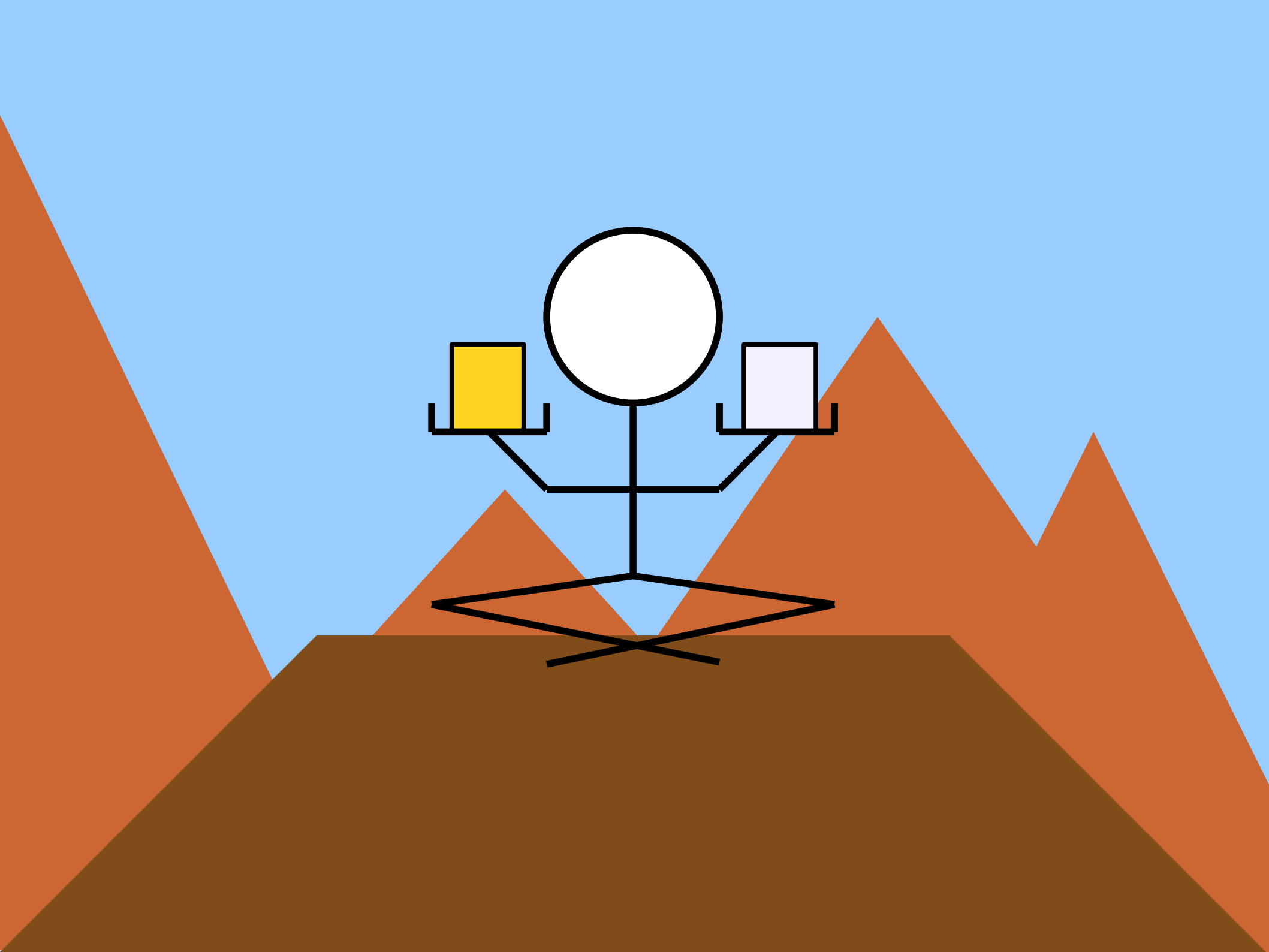
$$p \text{ loves } q$$

Since these relations focus on connections between two objects, they are called **binary relations**.

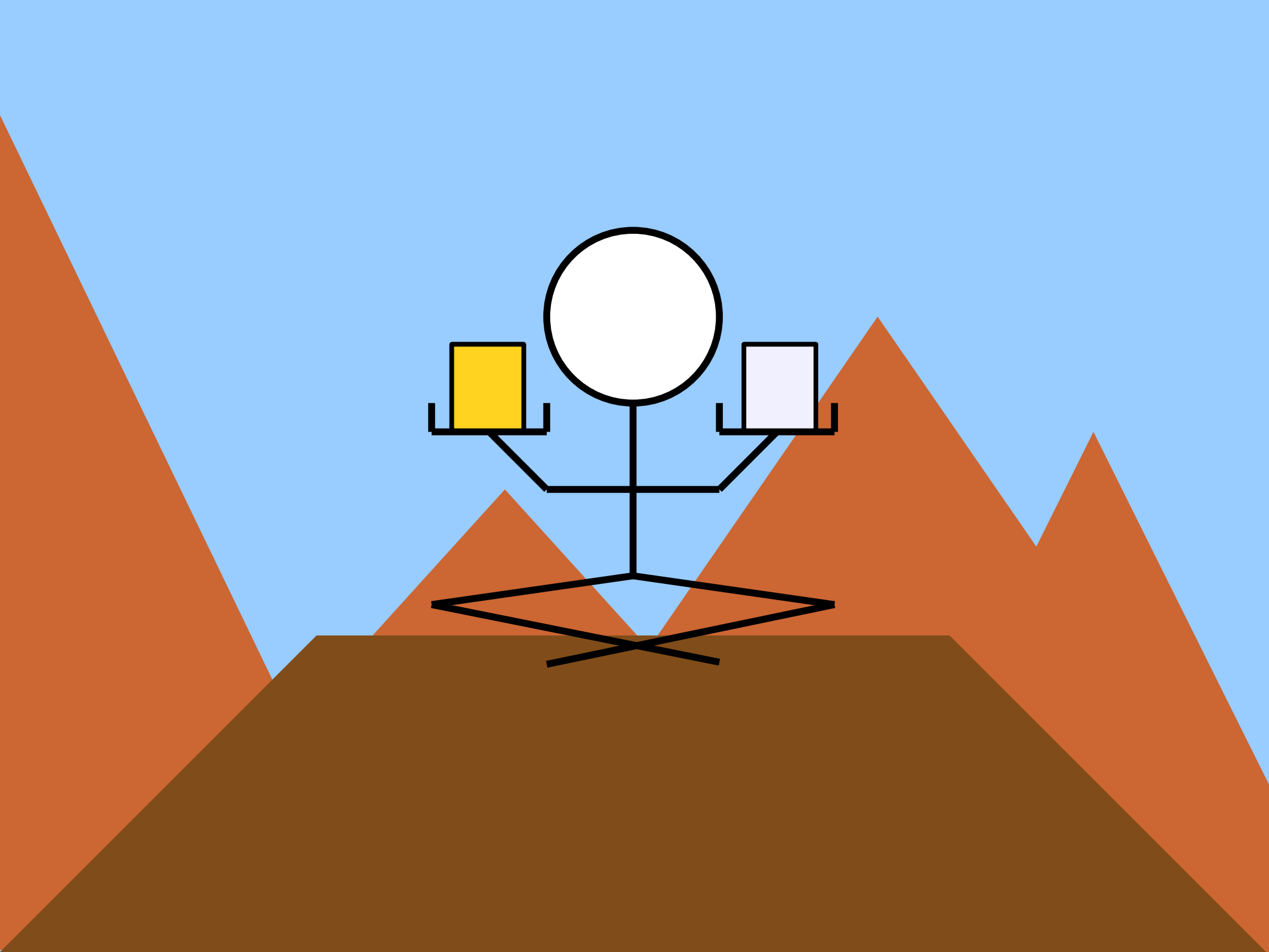
- The “binary” here means “pertaining to two things,” not “made of zeros and ones.”

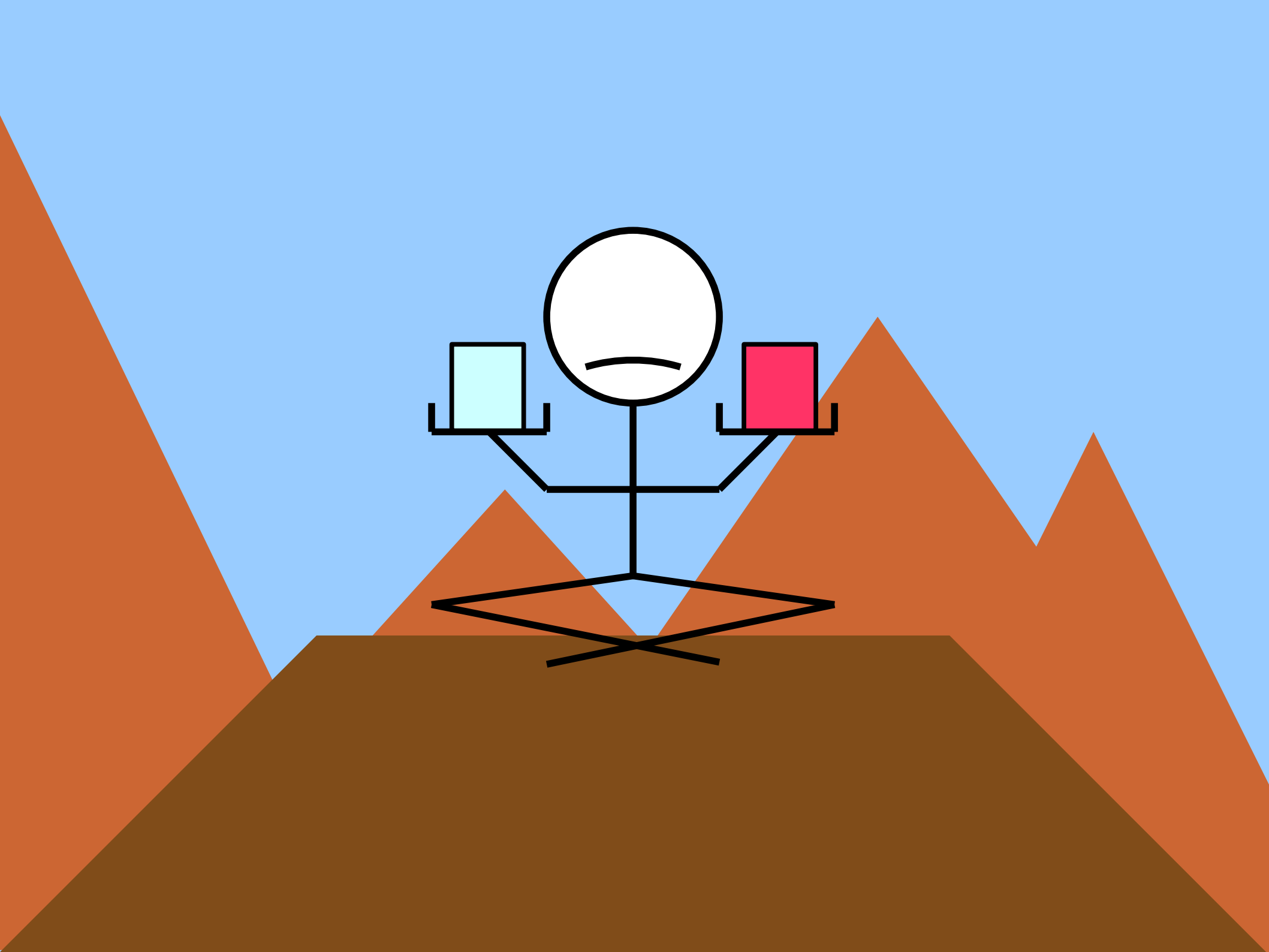
What exactly is a binary relation?

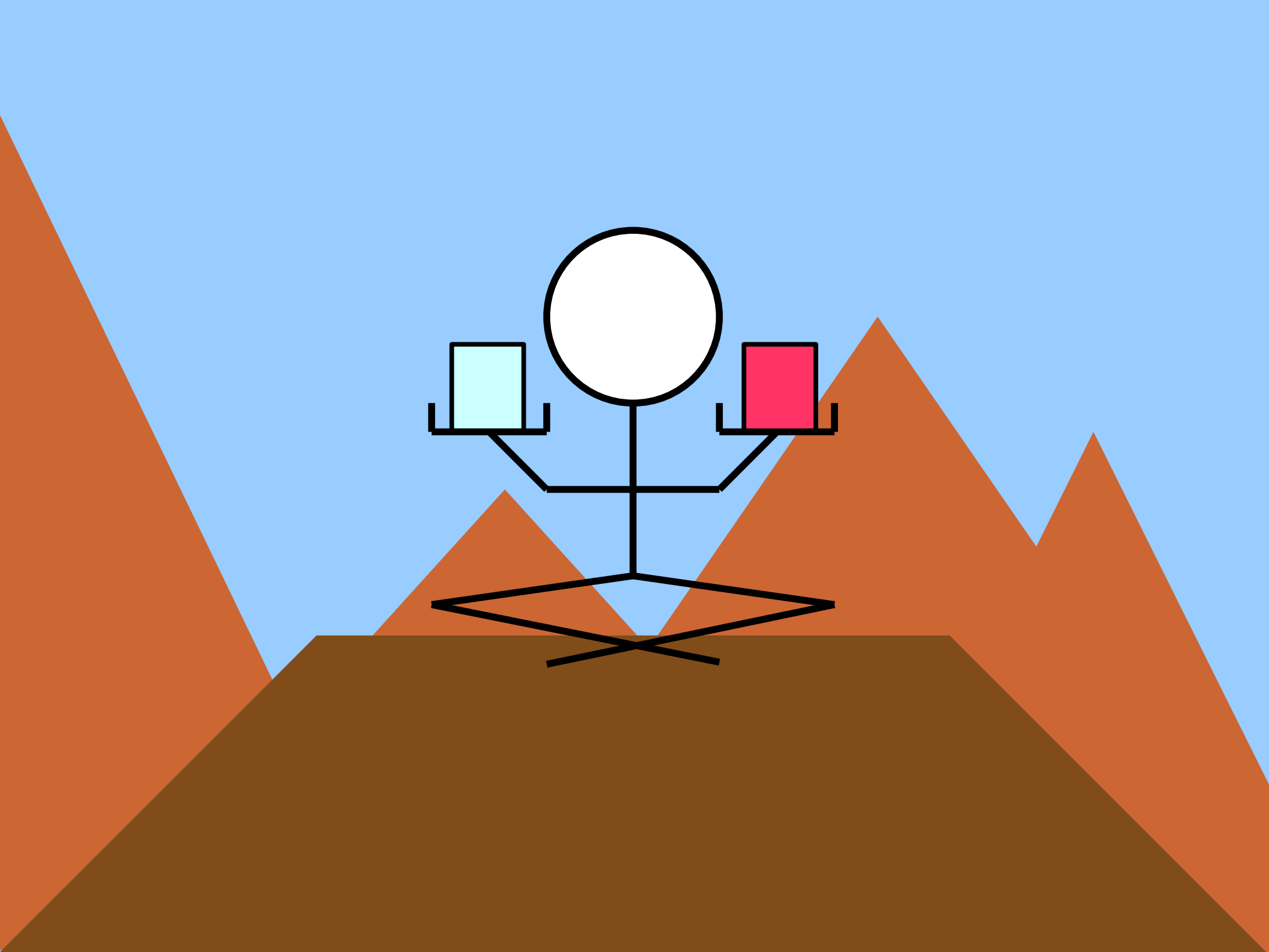




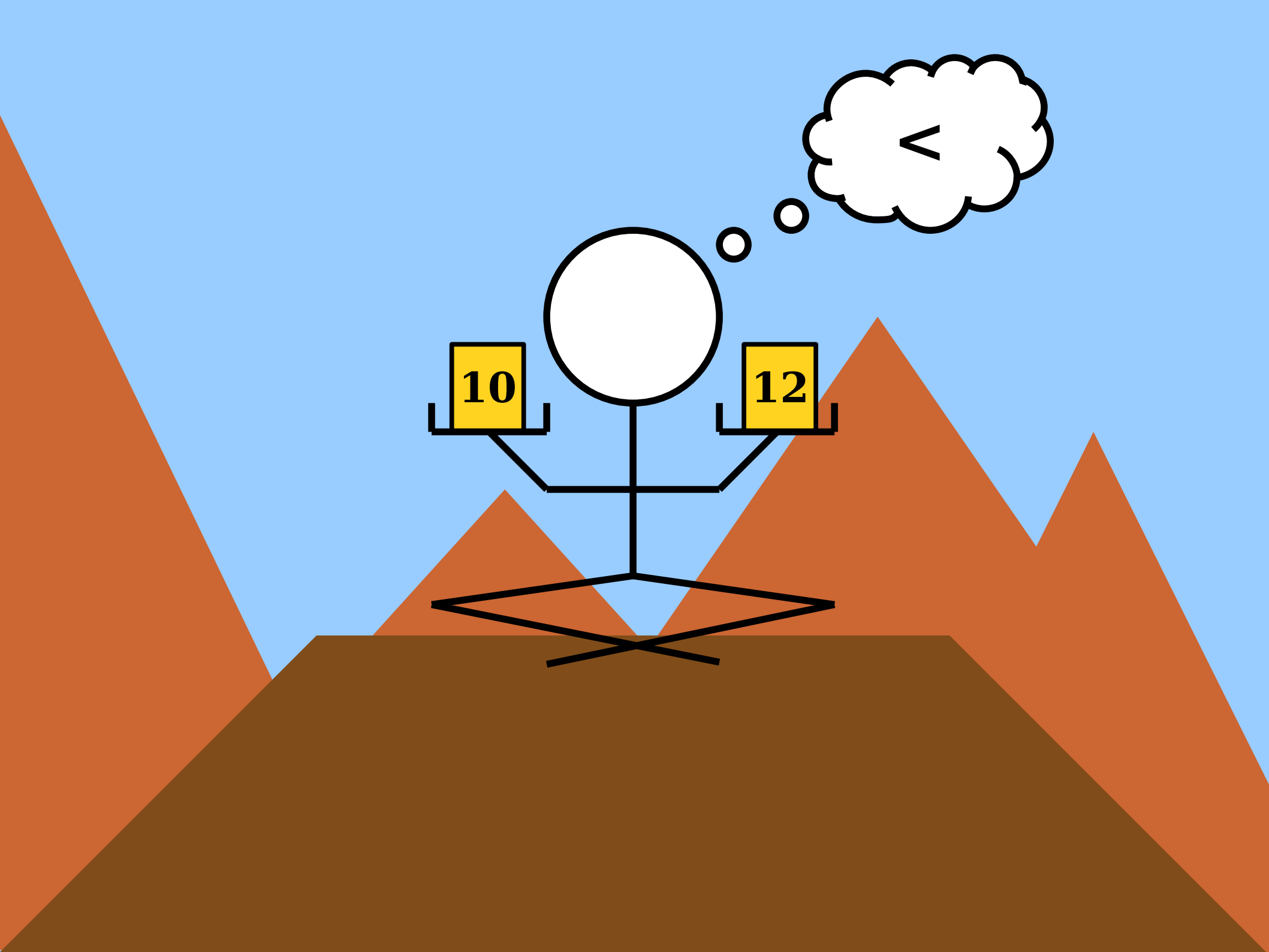








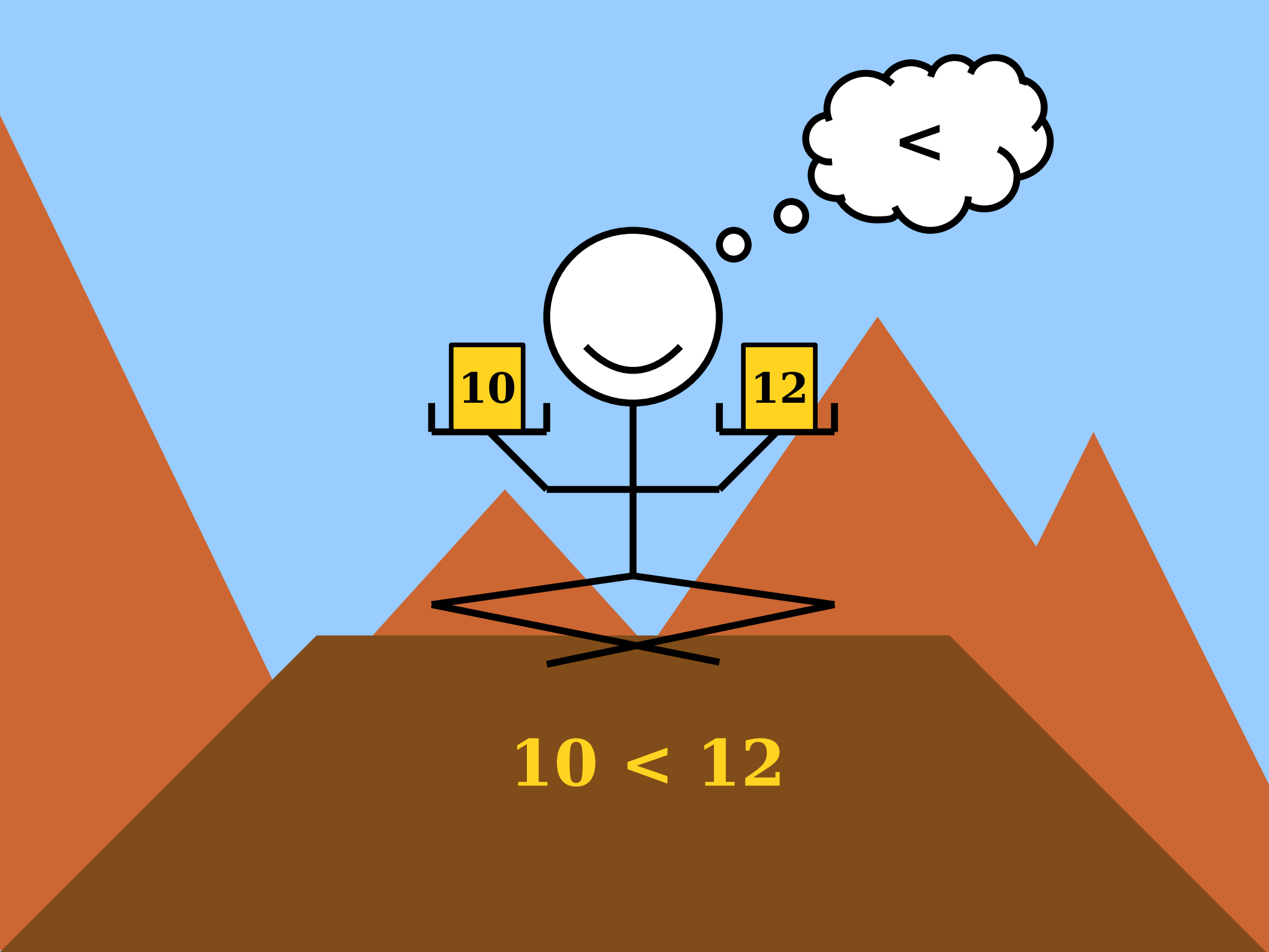




10

12

>



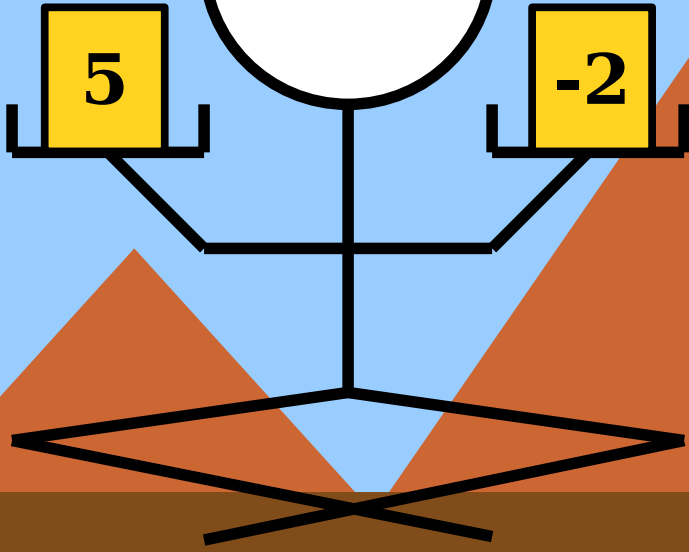
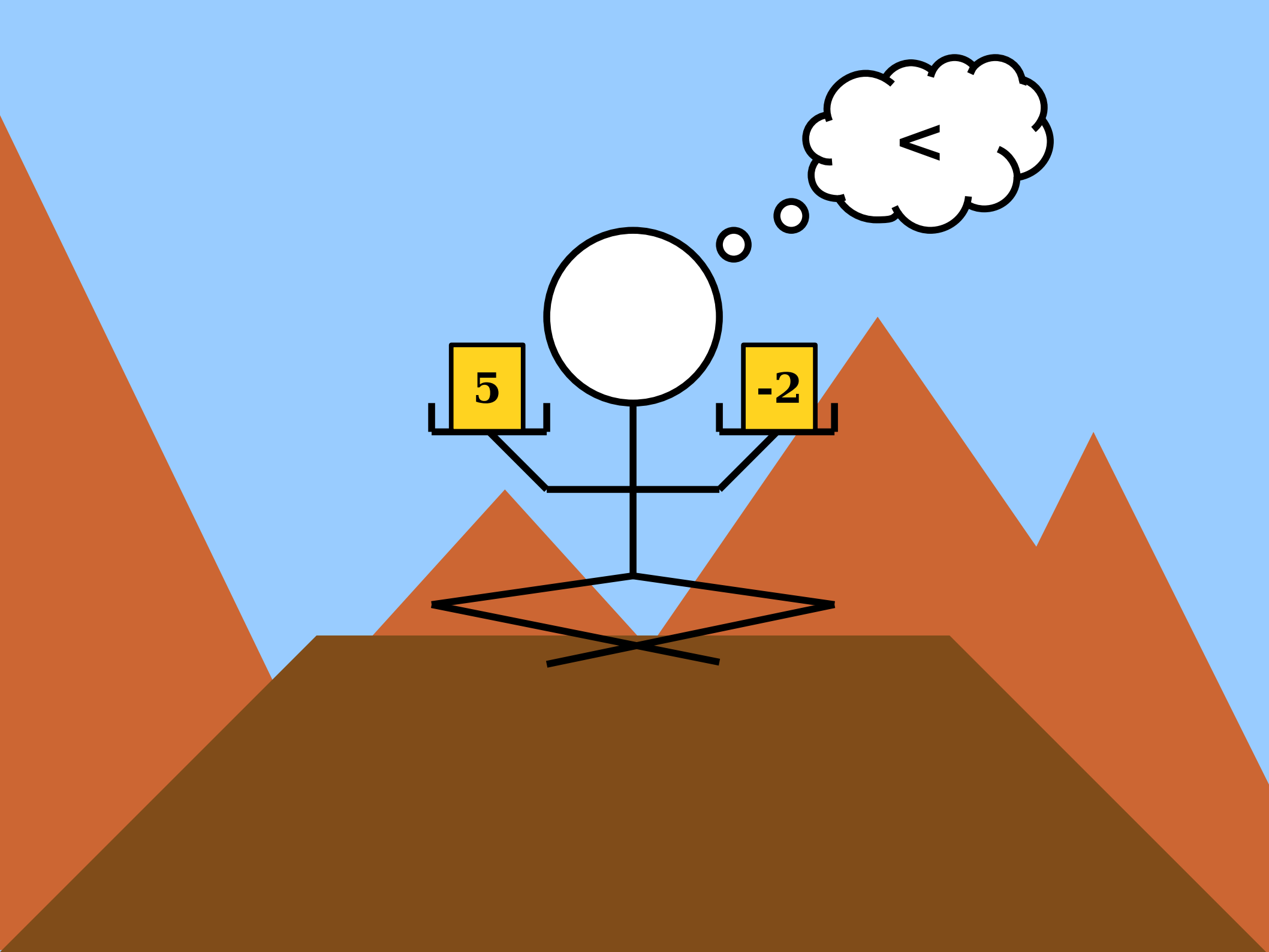
10

12

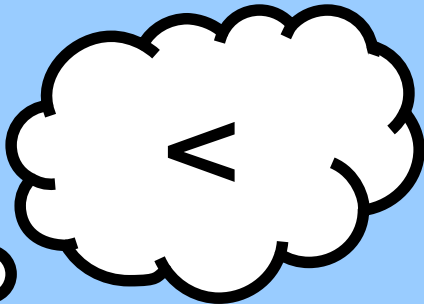
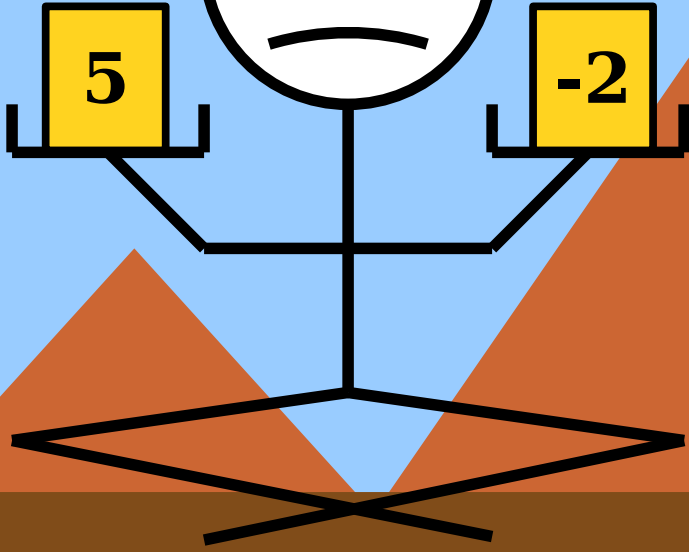
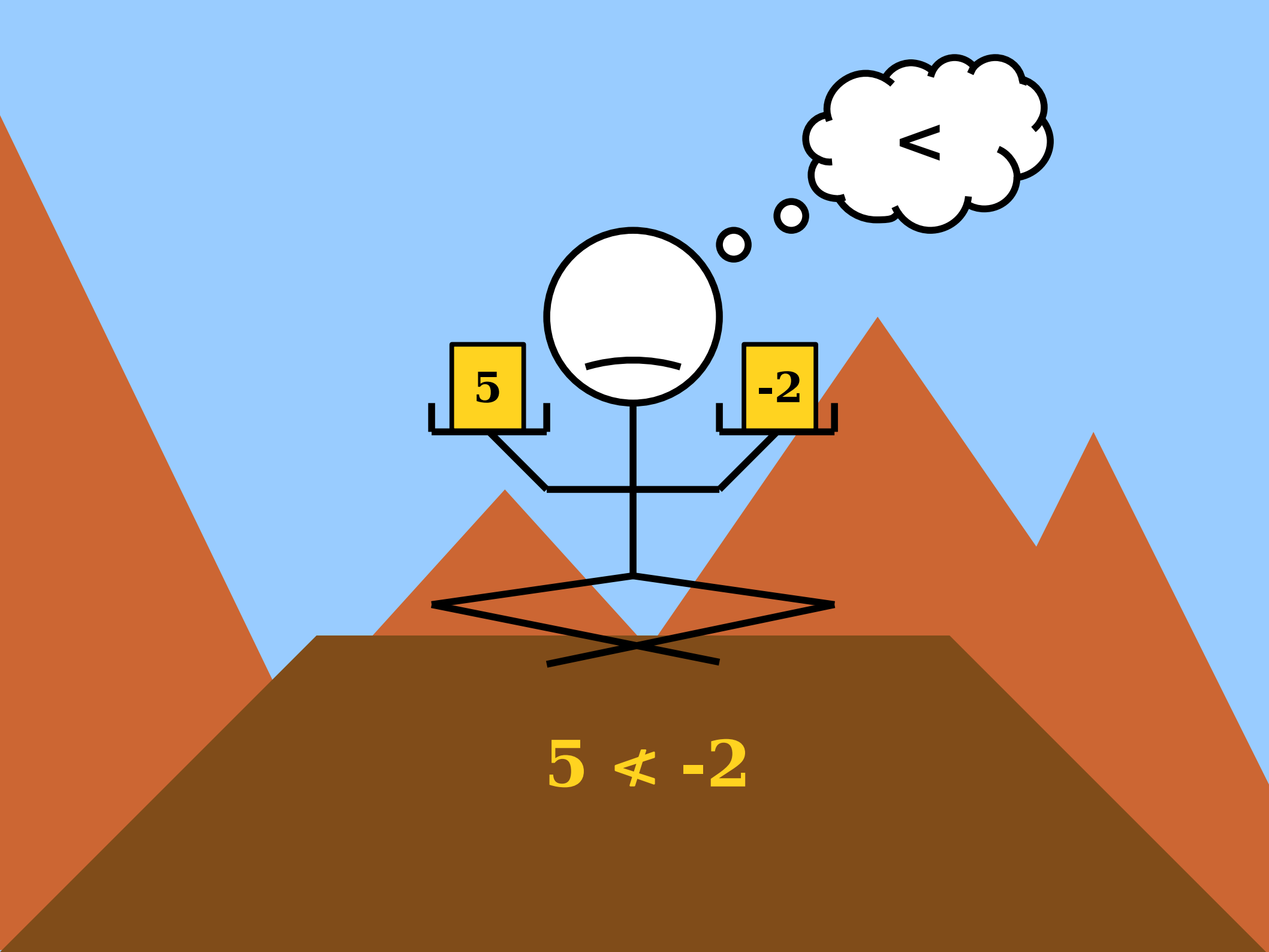
<

10 < 12

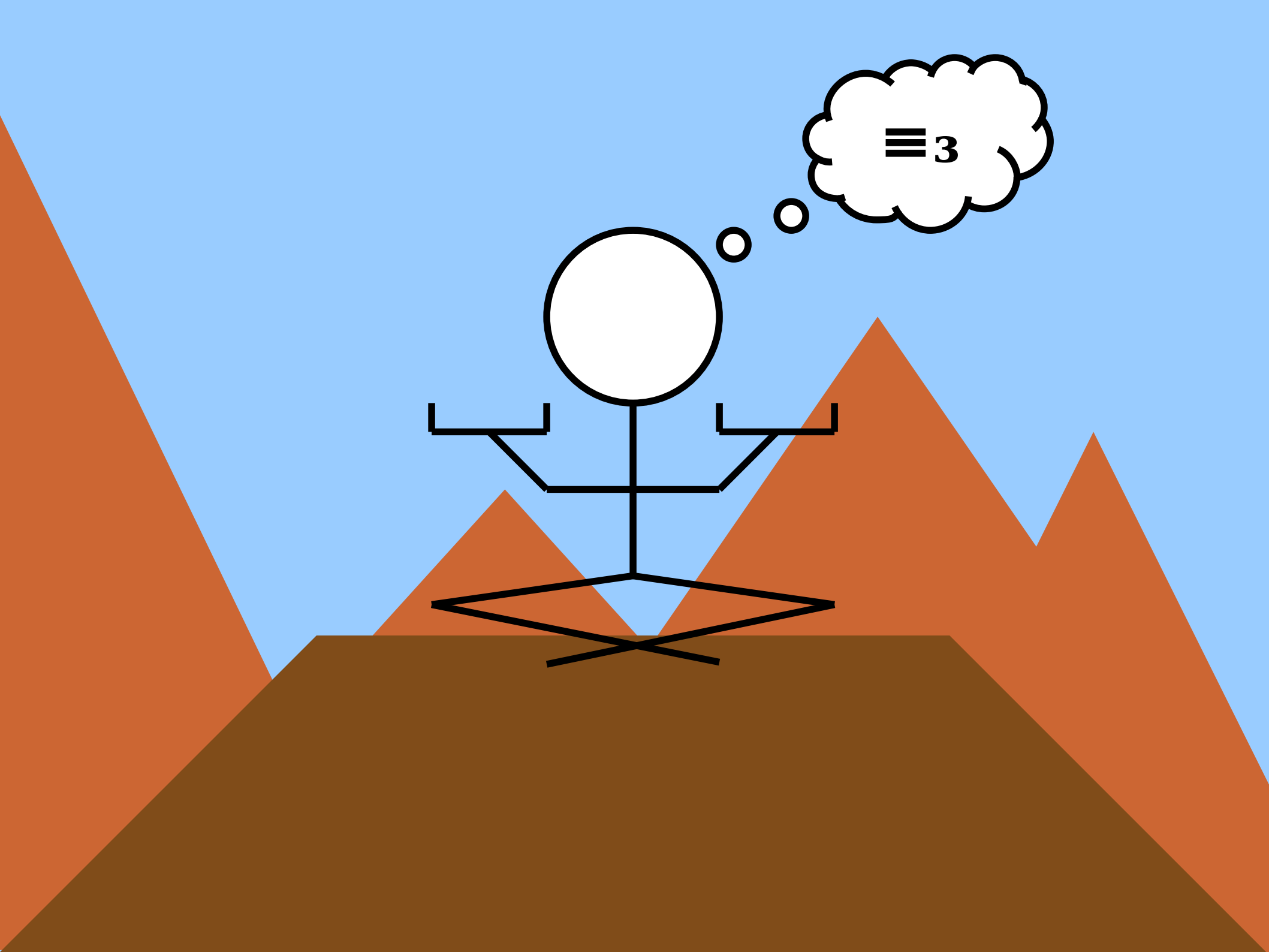




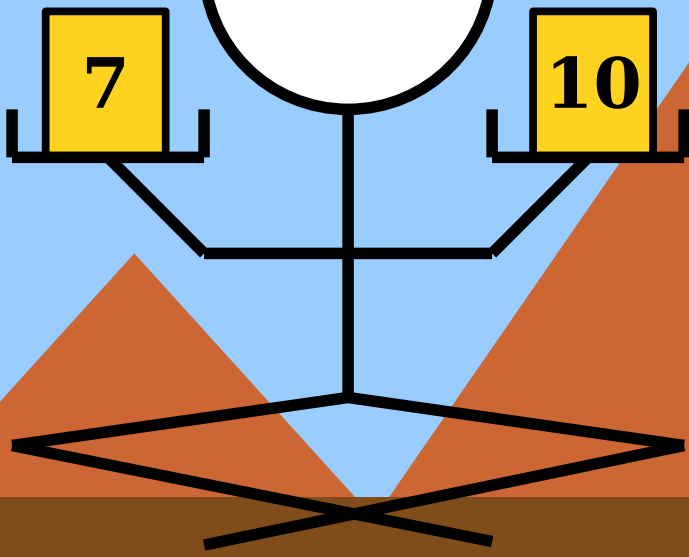
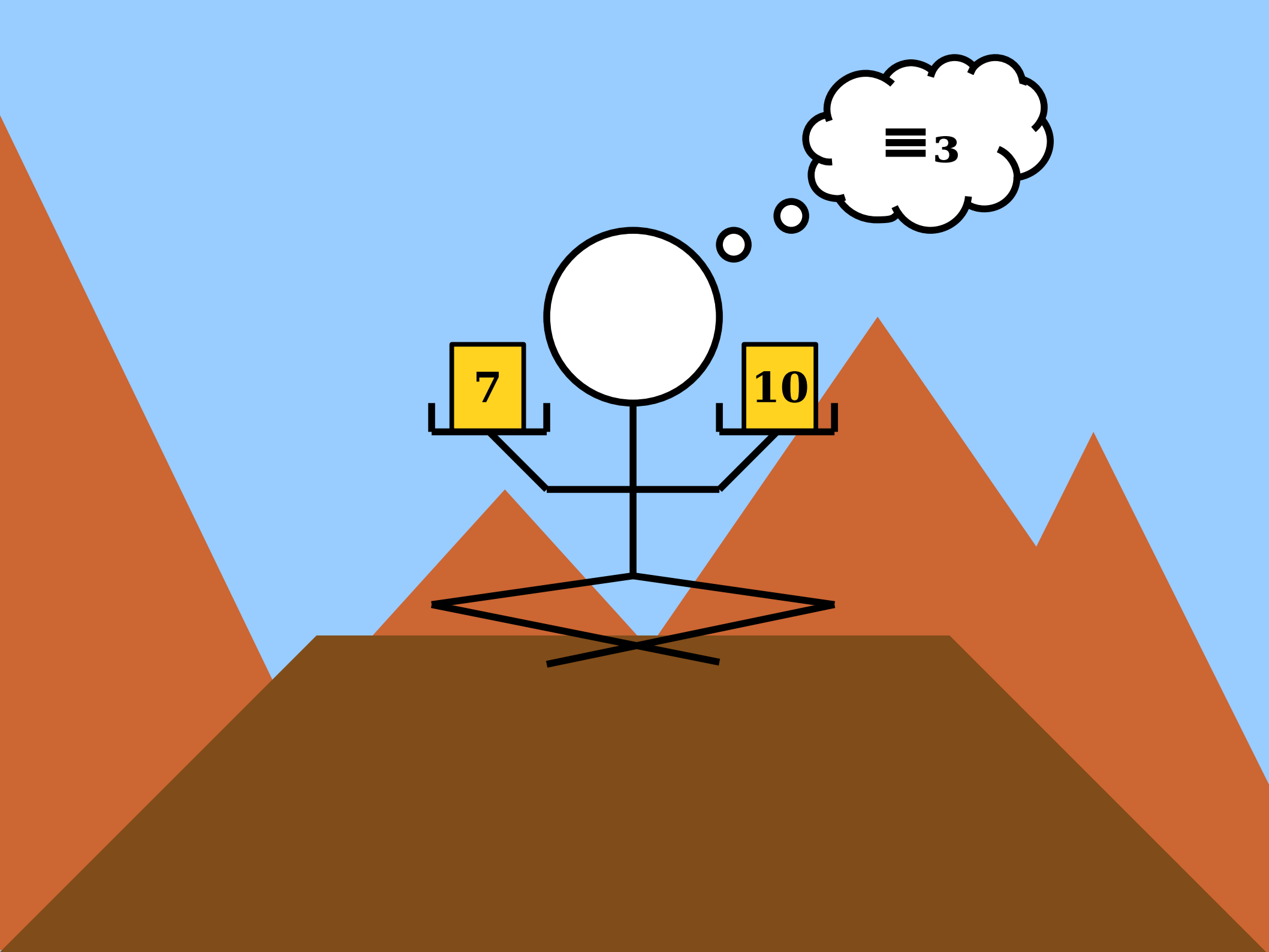
<



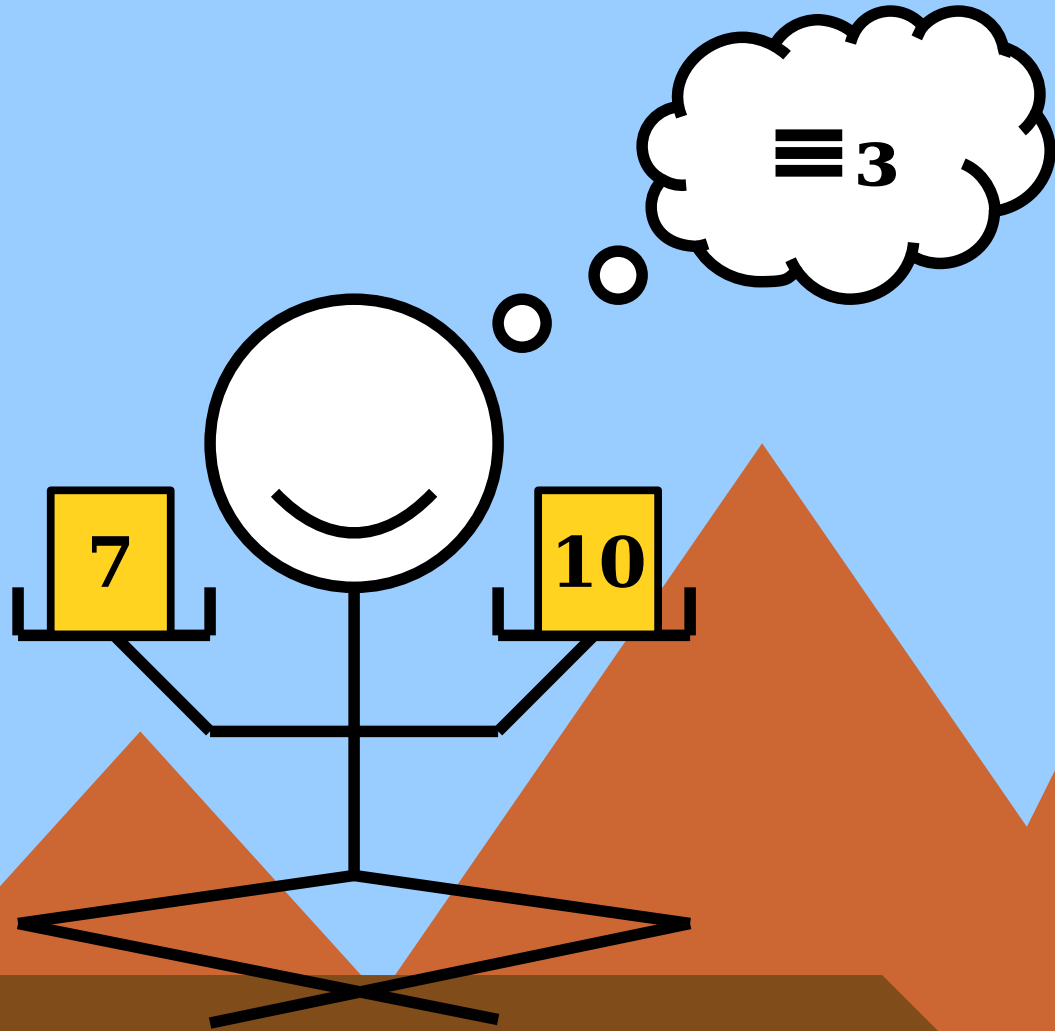
5 < -2



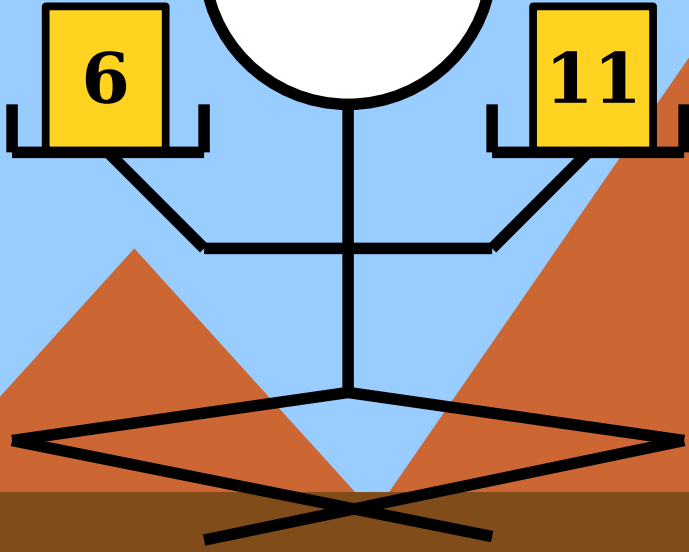
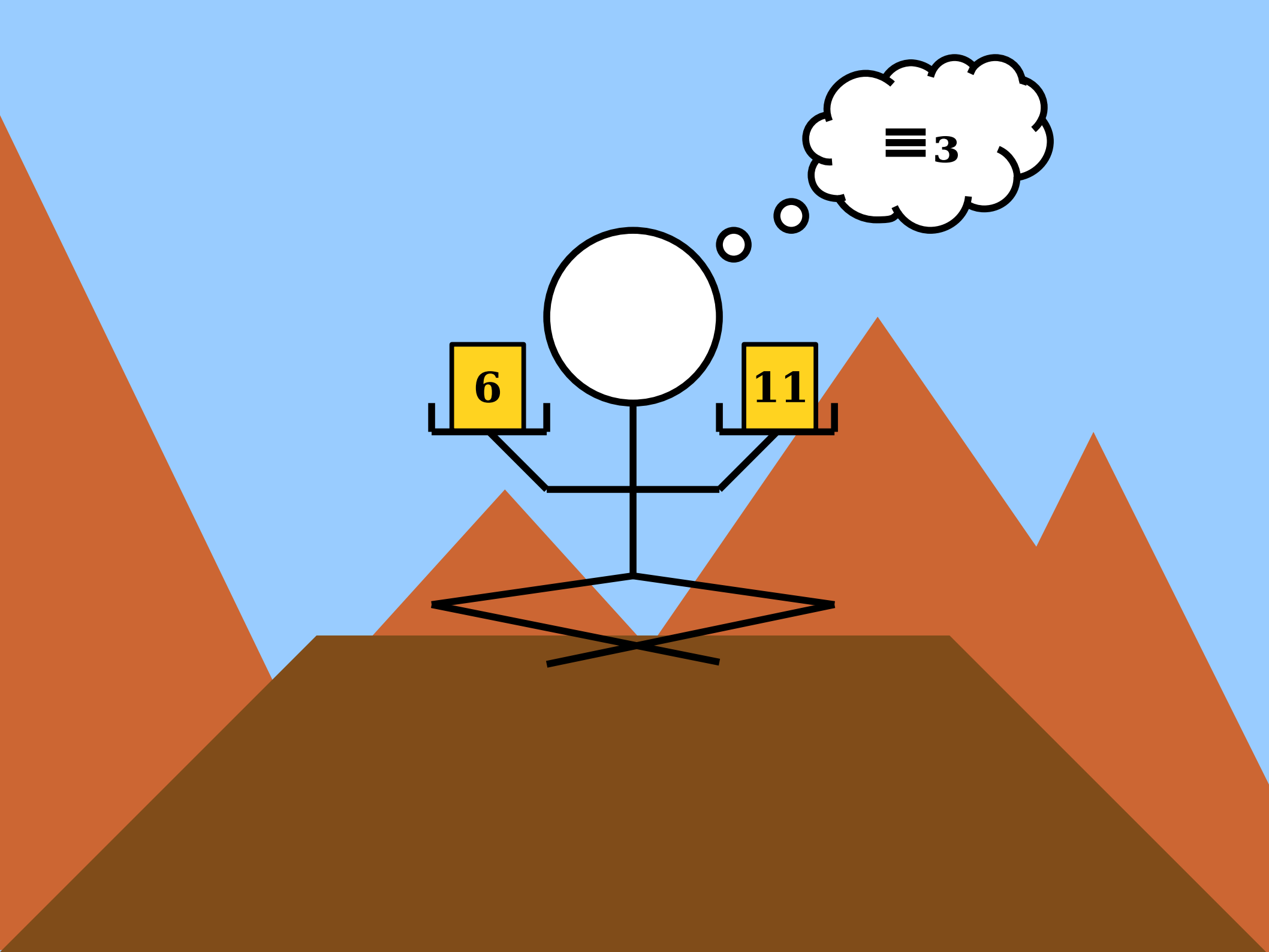
$$\equiv 3$$



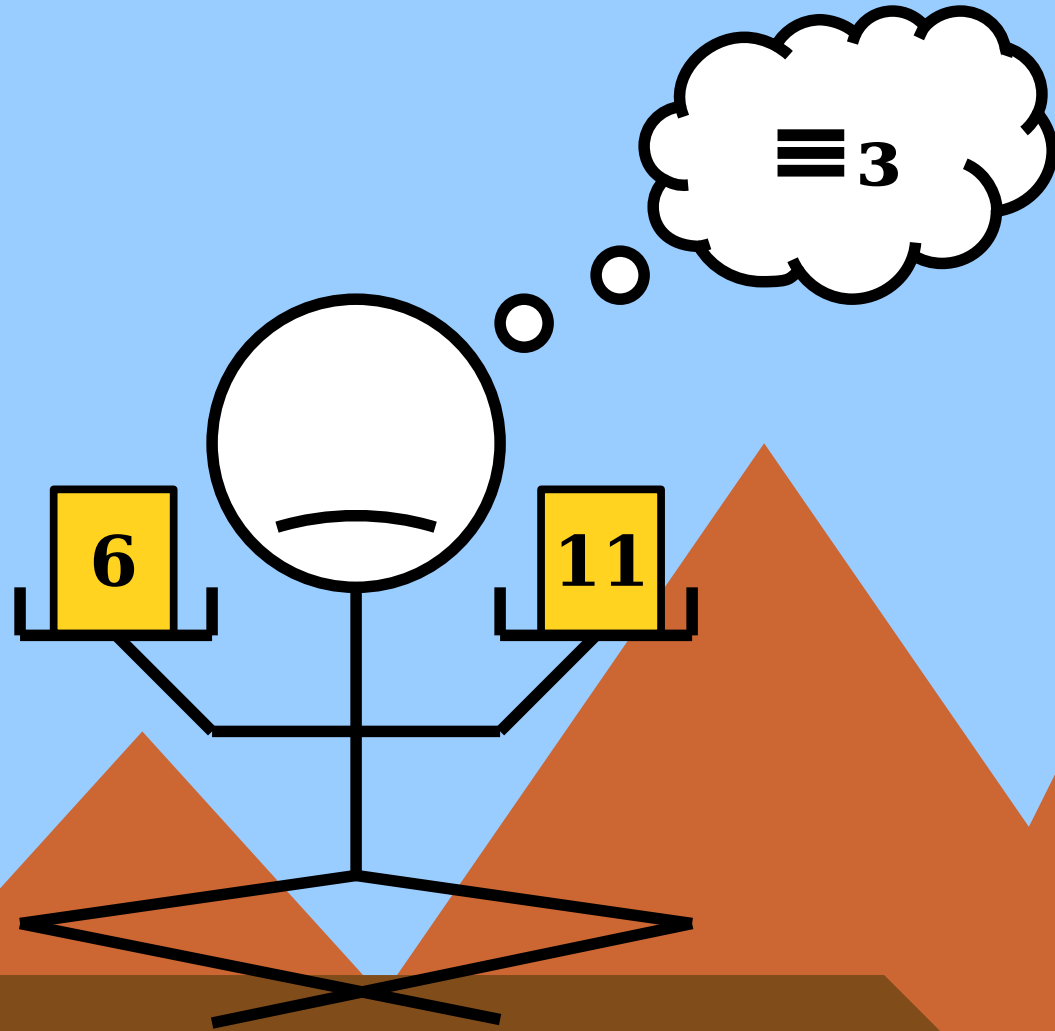
$3=3$



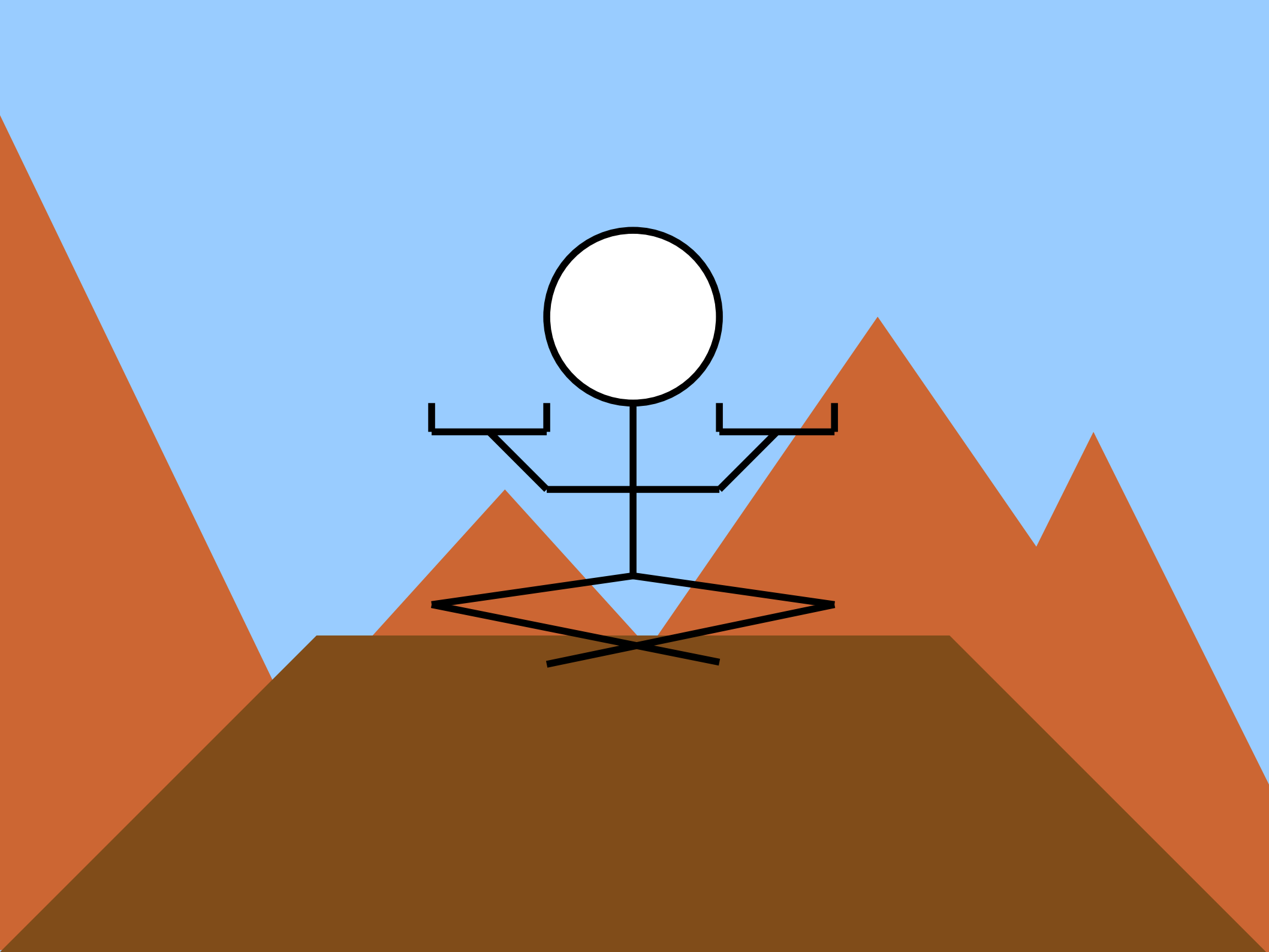
$$7 \equiv_3 10$$

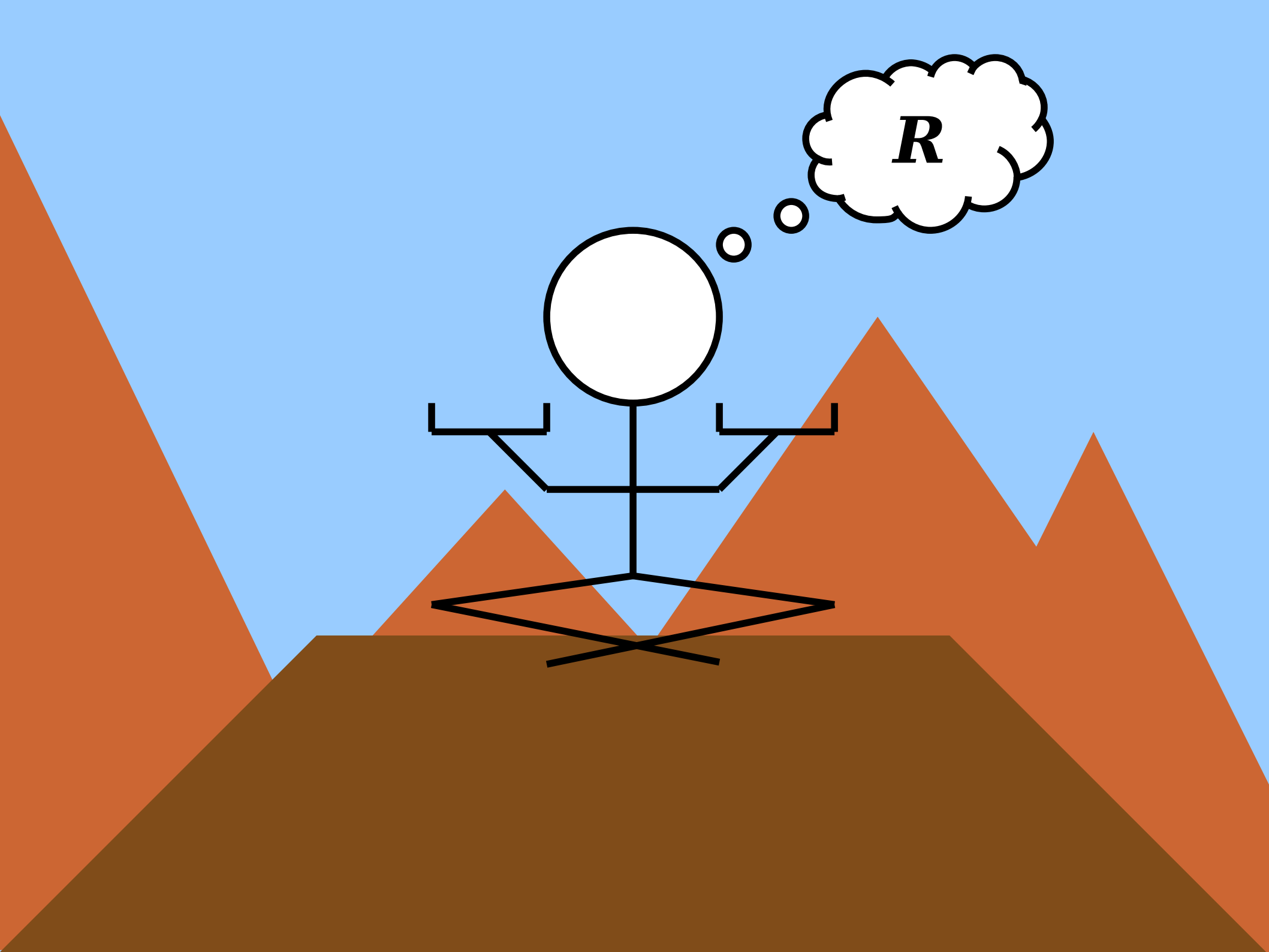


$3=3$

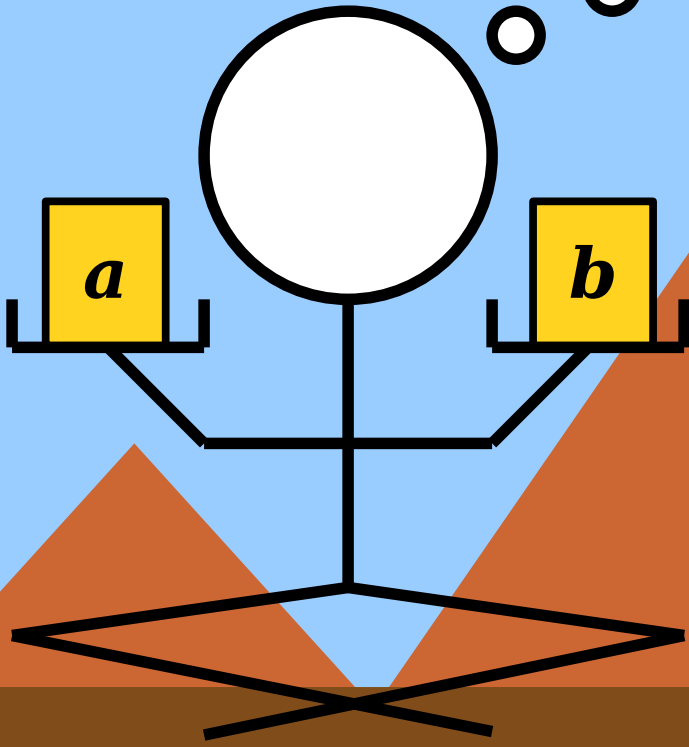
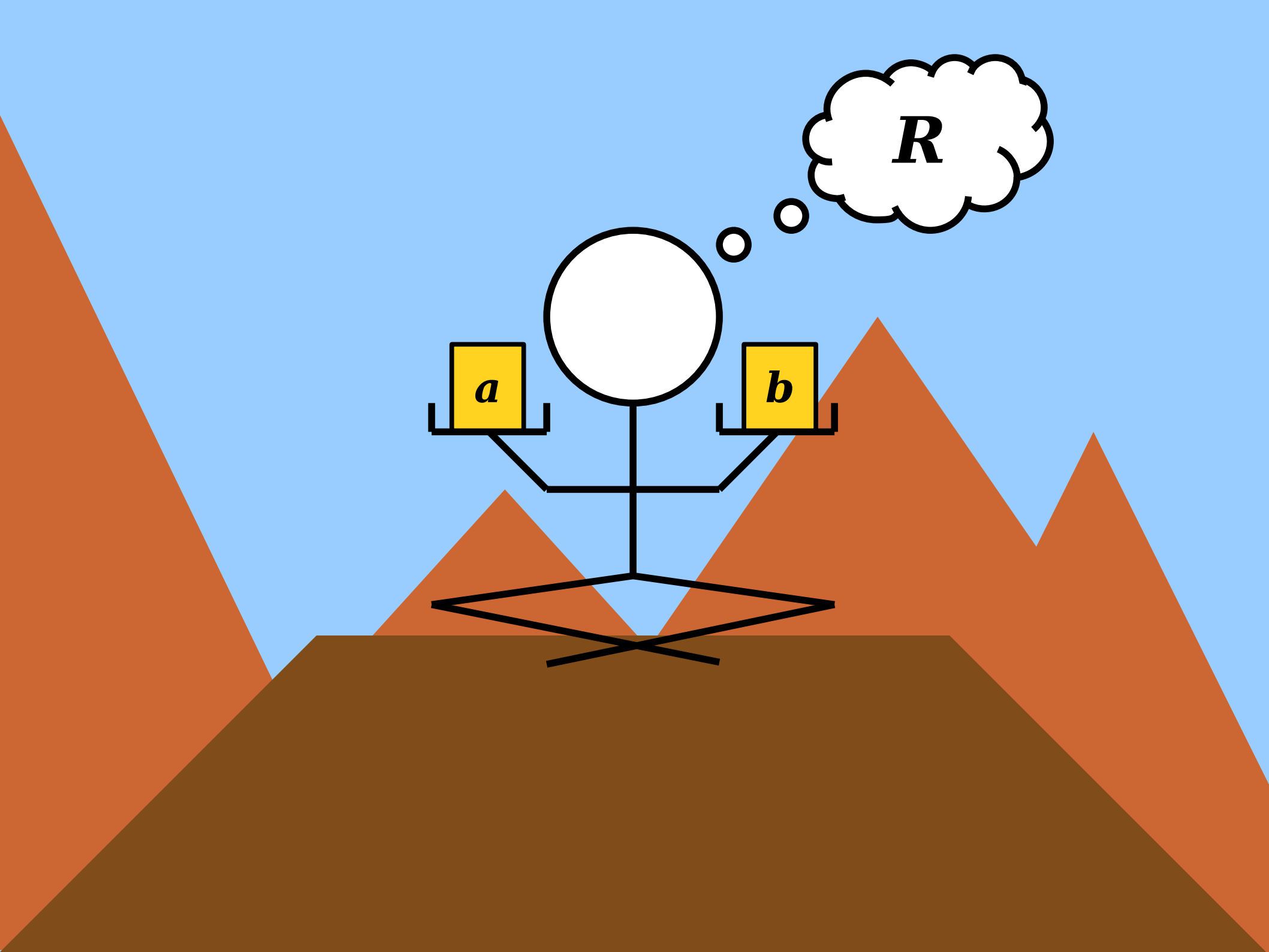


$$6 \not\equiv_3 11$$

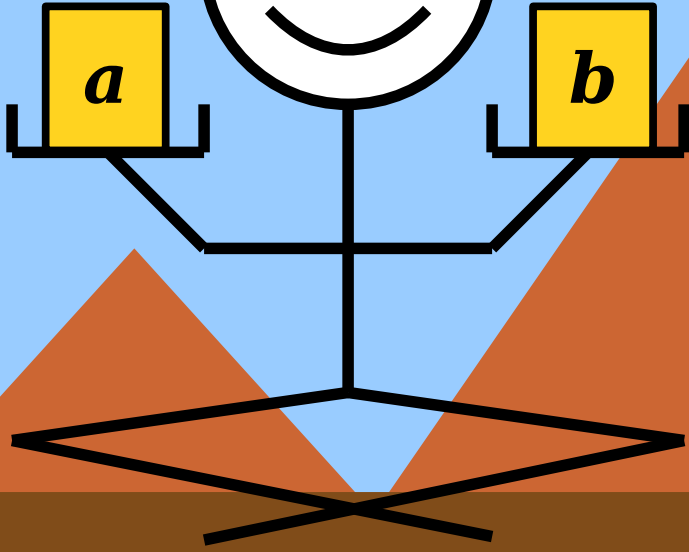
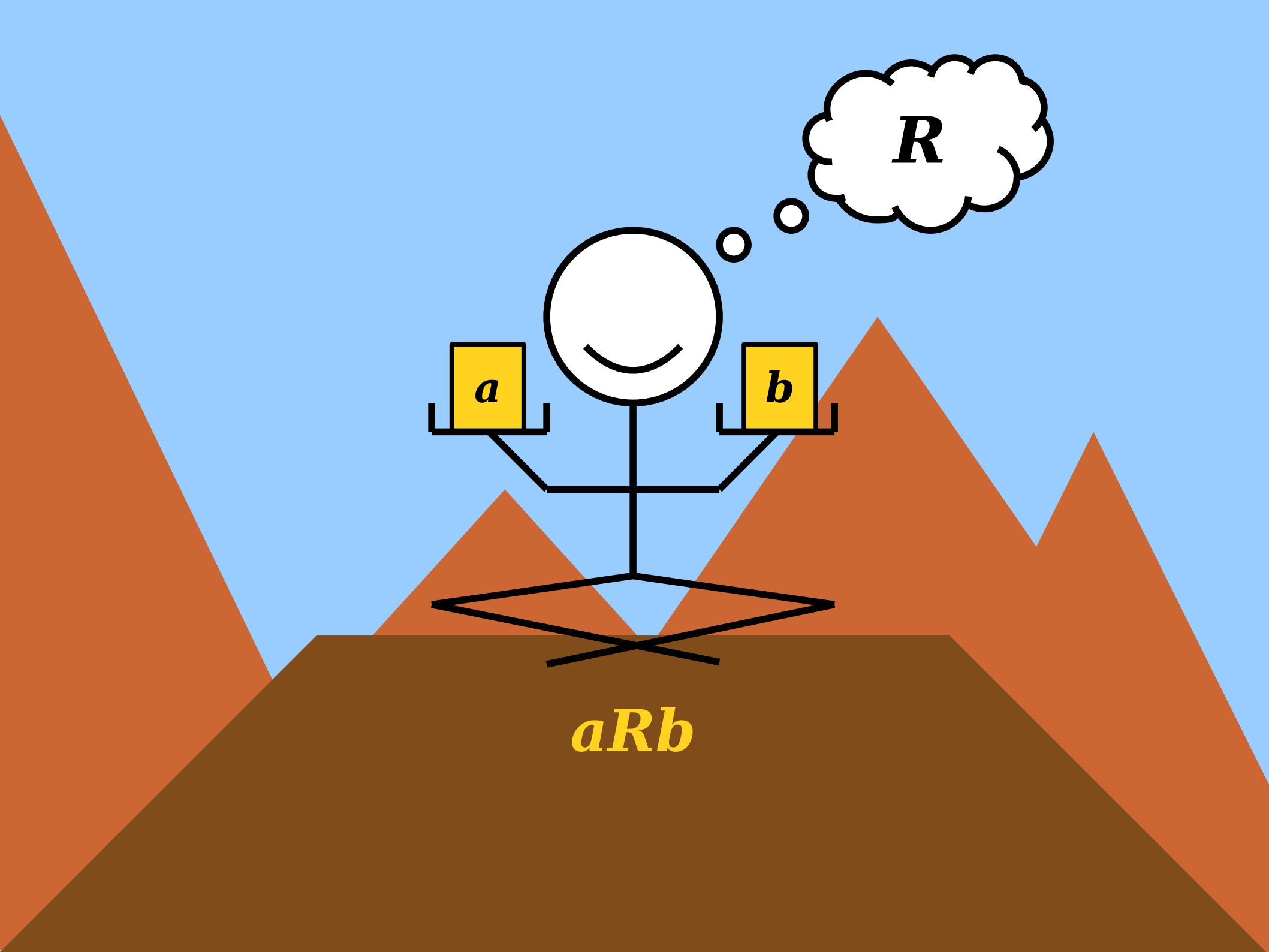




R

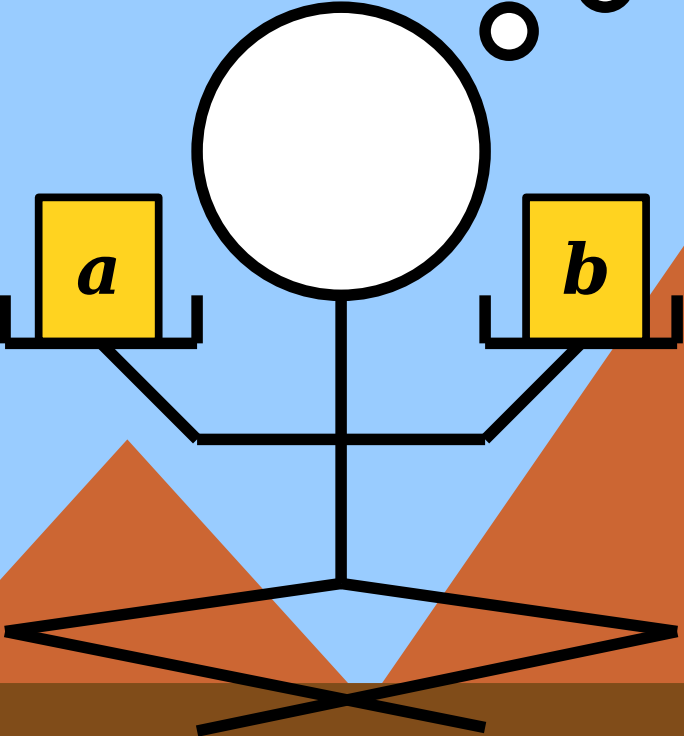
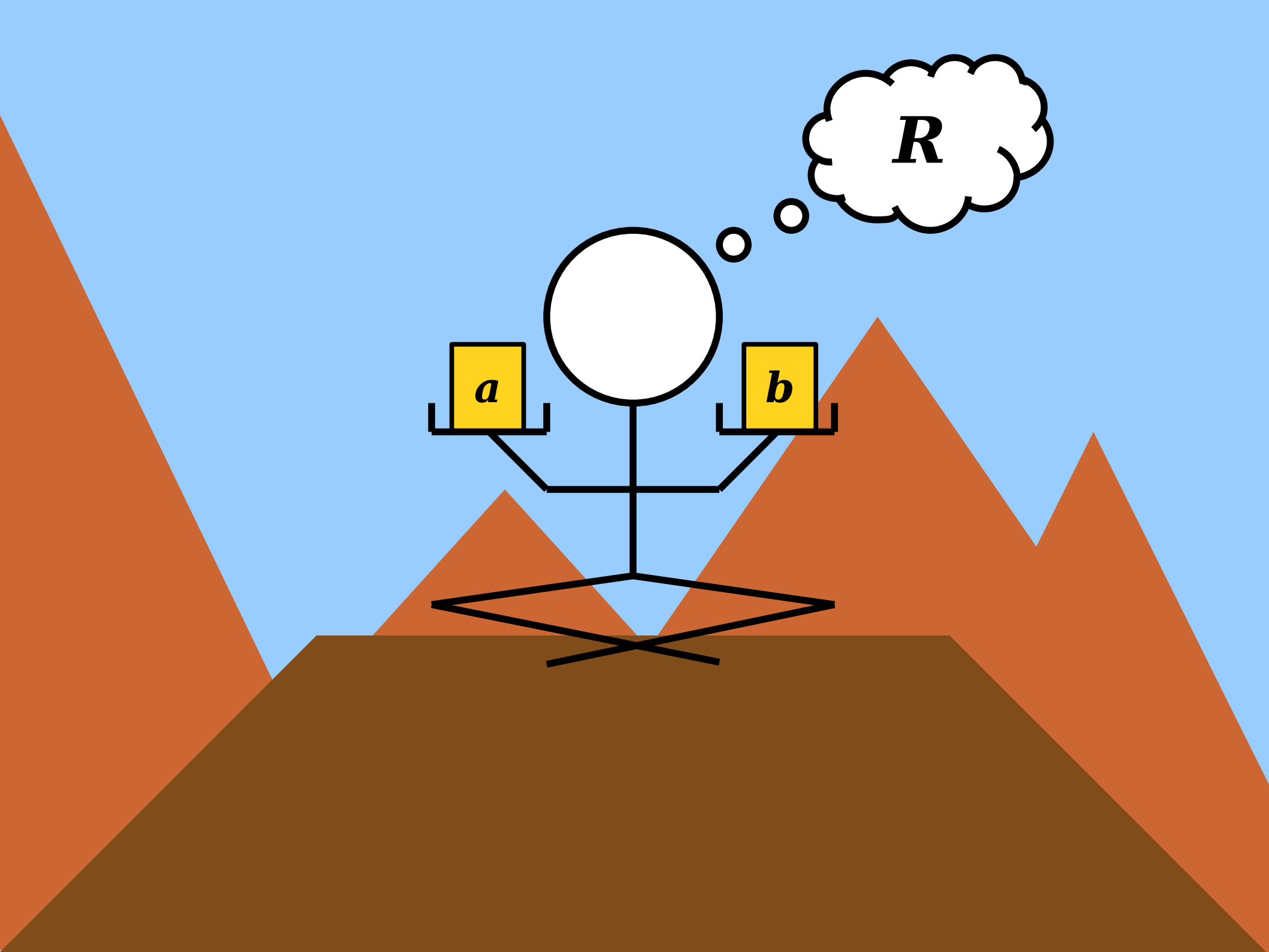


R



R

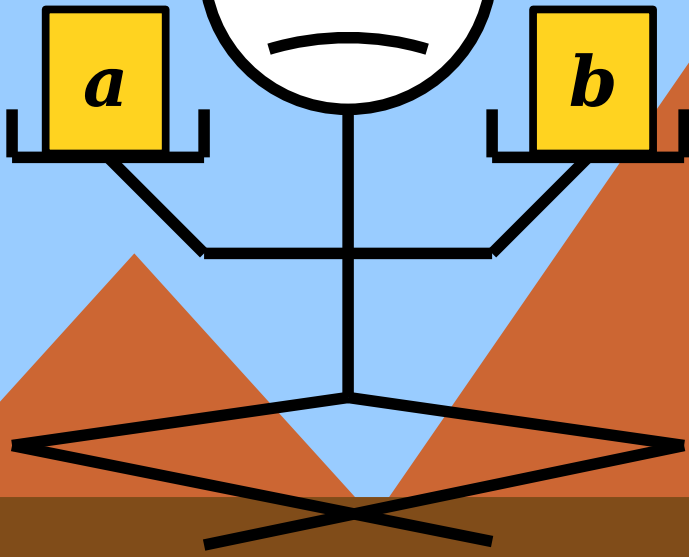
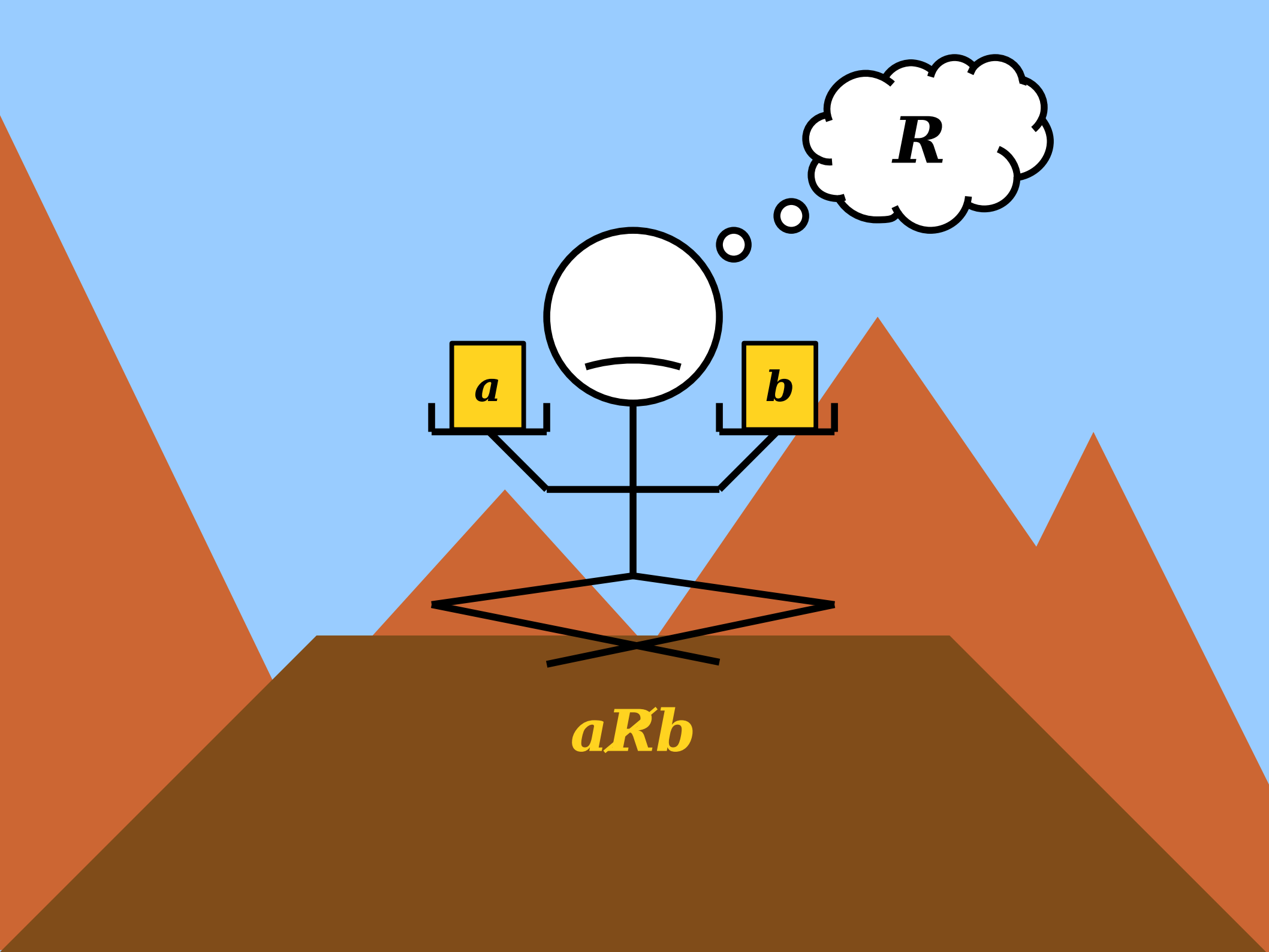
aRb



R

a

b



R

aRb

Binary Relations

A **binary relation over a set A** is a predicate R that can be applied to ordered pairs of elements drawn from A .

If R is a binary relation over A and it holds for the pair (a, b) , we write **aRb** .

$$3 = 3$$

$$5 < 7$$

$$\emptyset \subseteq \mathbb{N}$$

If R is a binary relation over A and it does not hold for the pair (a, b) , we write **$a \not R b$** .

$$4 \neq 3$$

$$4 \not < 3$$

$$\mathbb{N} \not \subseteq \emptyset$$

Properties of Relations

Generally speaking, if R is a binary relation over a set A , the order of the operands is significant.

- For example, $3 < 5$, but $5 \not< 3$.

In some relations order is irrelevant; more on that later.

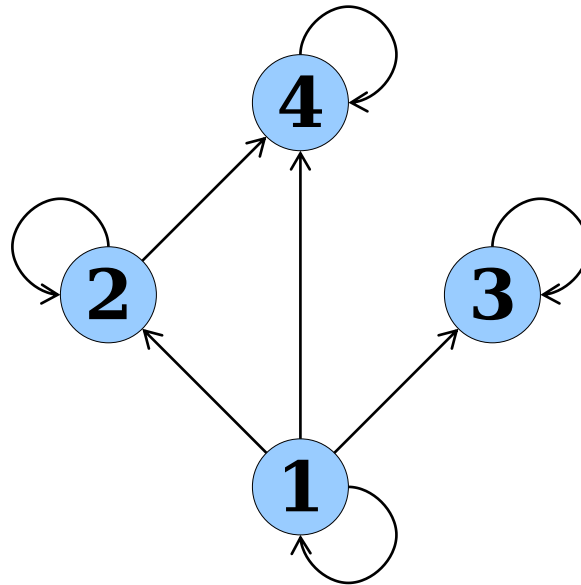
Relations are always defined relative to some underlying set.

It's not meaningful to ask whether $\odot \subseteq 15$, for example, since \subseteq is defined over sets, not arbitrary objects.

Visualizing Relations

We can visualize a binary relation R over a set A by drawing the elements of A and drawing an arrow between an element a and an element b if aRb is true.

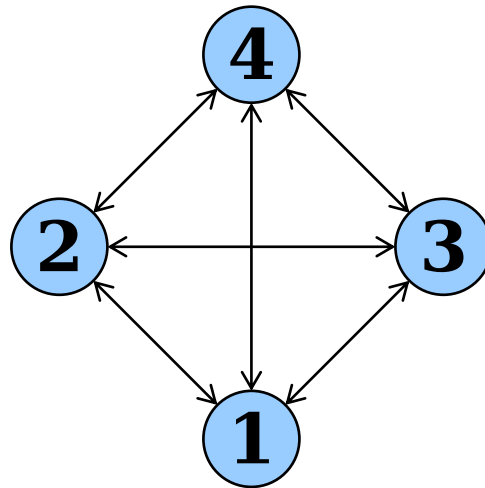
Example: the relation $a \mid b$ (meaning “ a divides b ”) over the set $\{1, 2, 3, 4\}$ looks like this:



Visualizing Relations

We can visualize a binary relation R over a set A by drawing the elements of A and drawing an arrow between an element a and an element b if aRb is true.

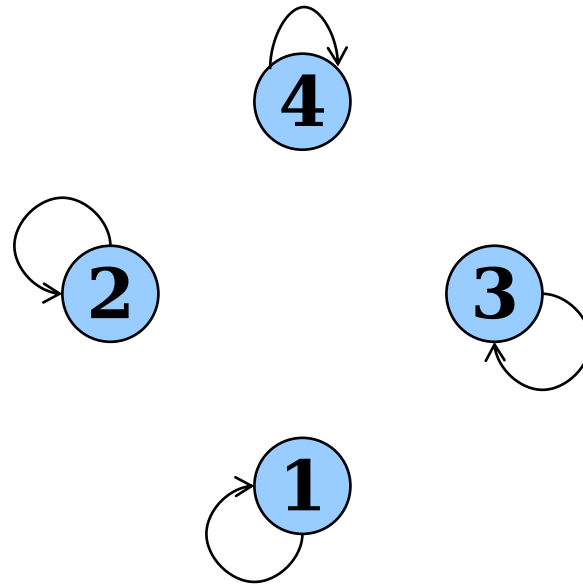
Example: the relation $a \neq b$ over the set $\{1, 2, 3, 4\}$ looks like this:



Visualizing Relations

We can visualize a binary relation R over a set A by drawing the elements of A and drawing an arrow between an element a and an element b if aRb is true.

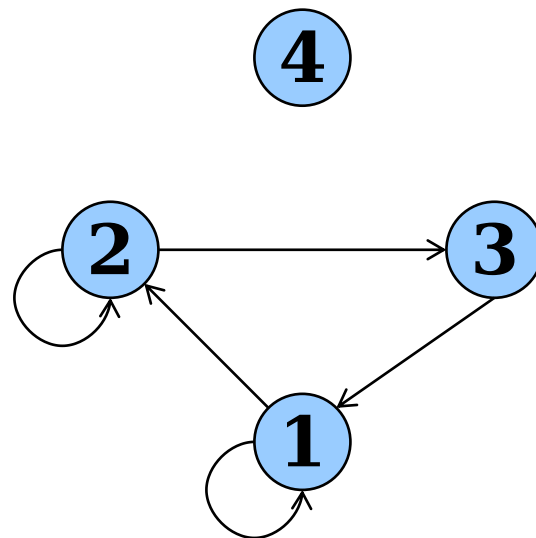
Example: the relation $a = b$ over the set $\{1, 2, 3, 4\}$ looks like this:



Visualizing Relations

We can visualize a binary relation R over a set A by drawing the elements of A and drawing an arrow between an element a and an element b if aRb is true.

Example: below is some relation over $\{1, 2, 3, 4\}$ that's a totally valid relation even though there doesn't appear to be a simple unifying rule.

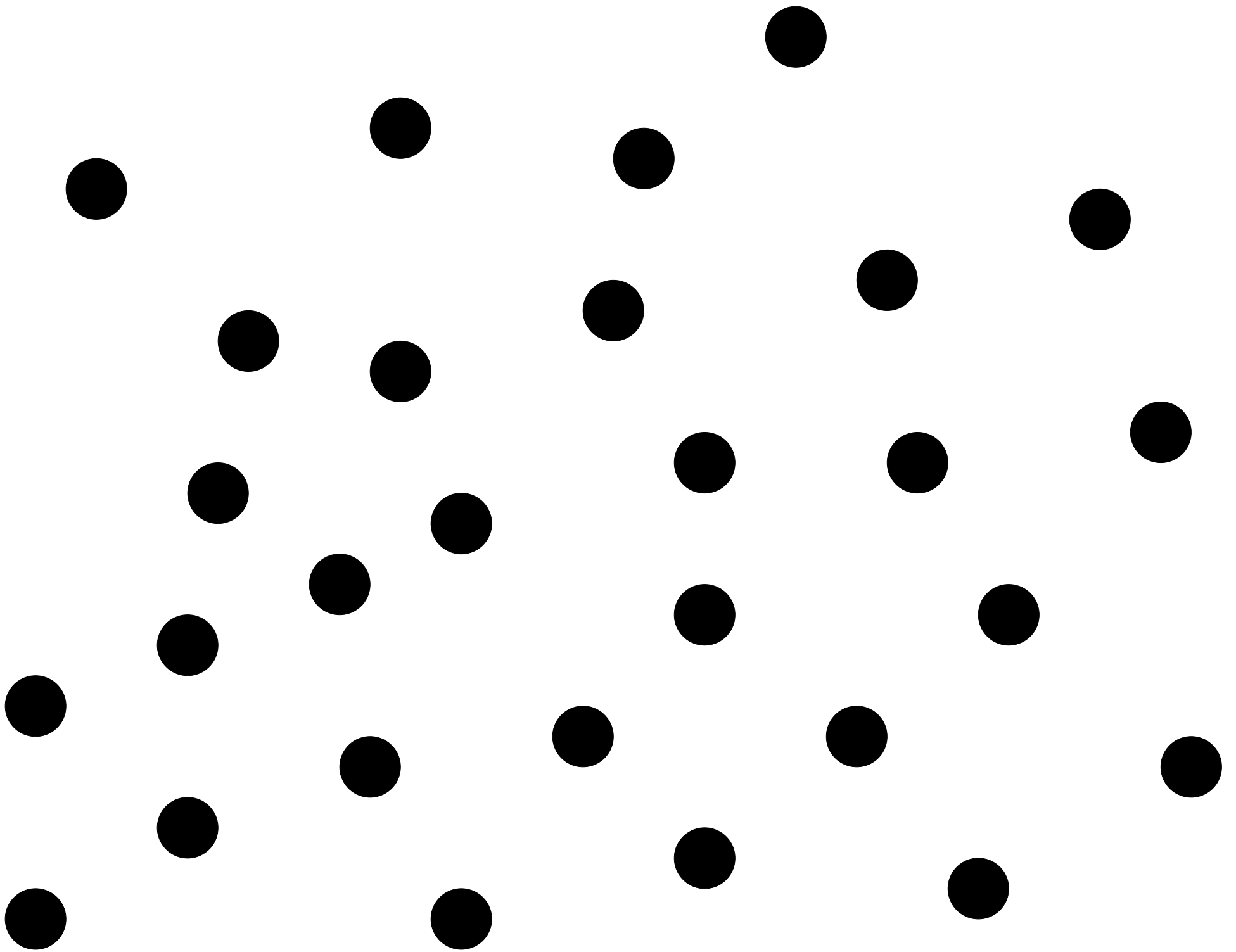


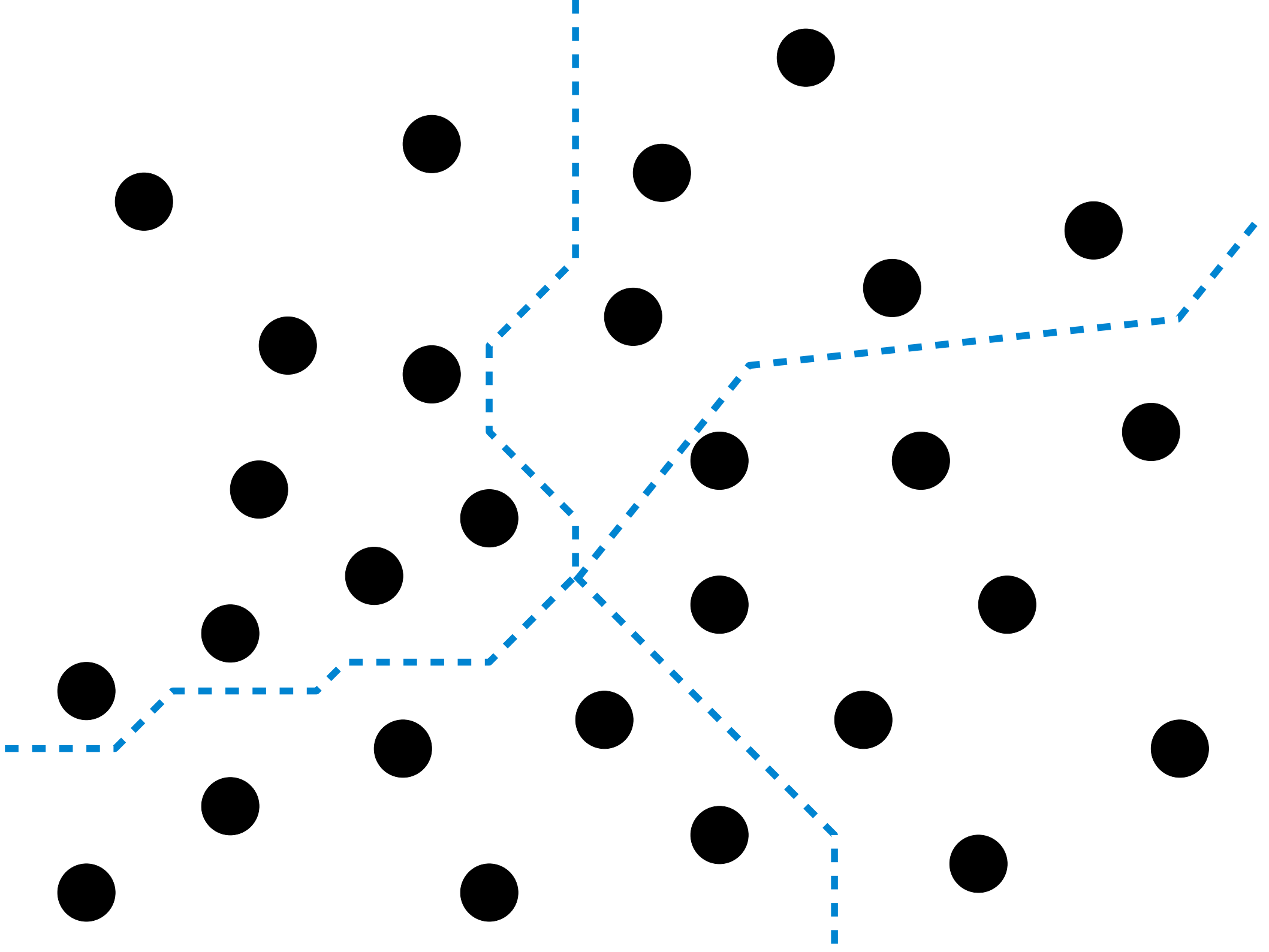
Capturing Structure

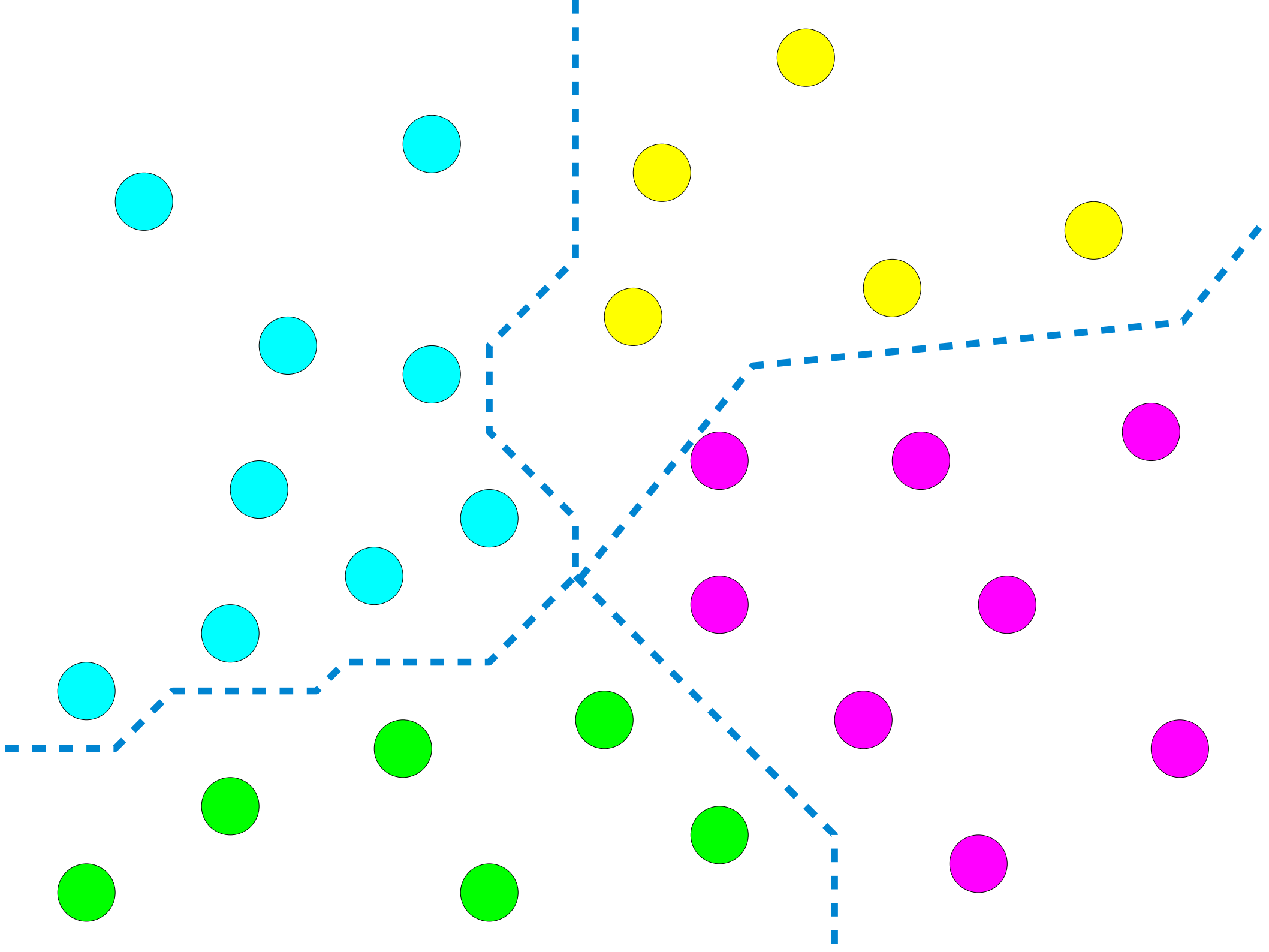
Capturing Structure

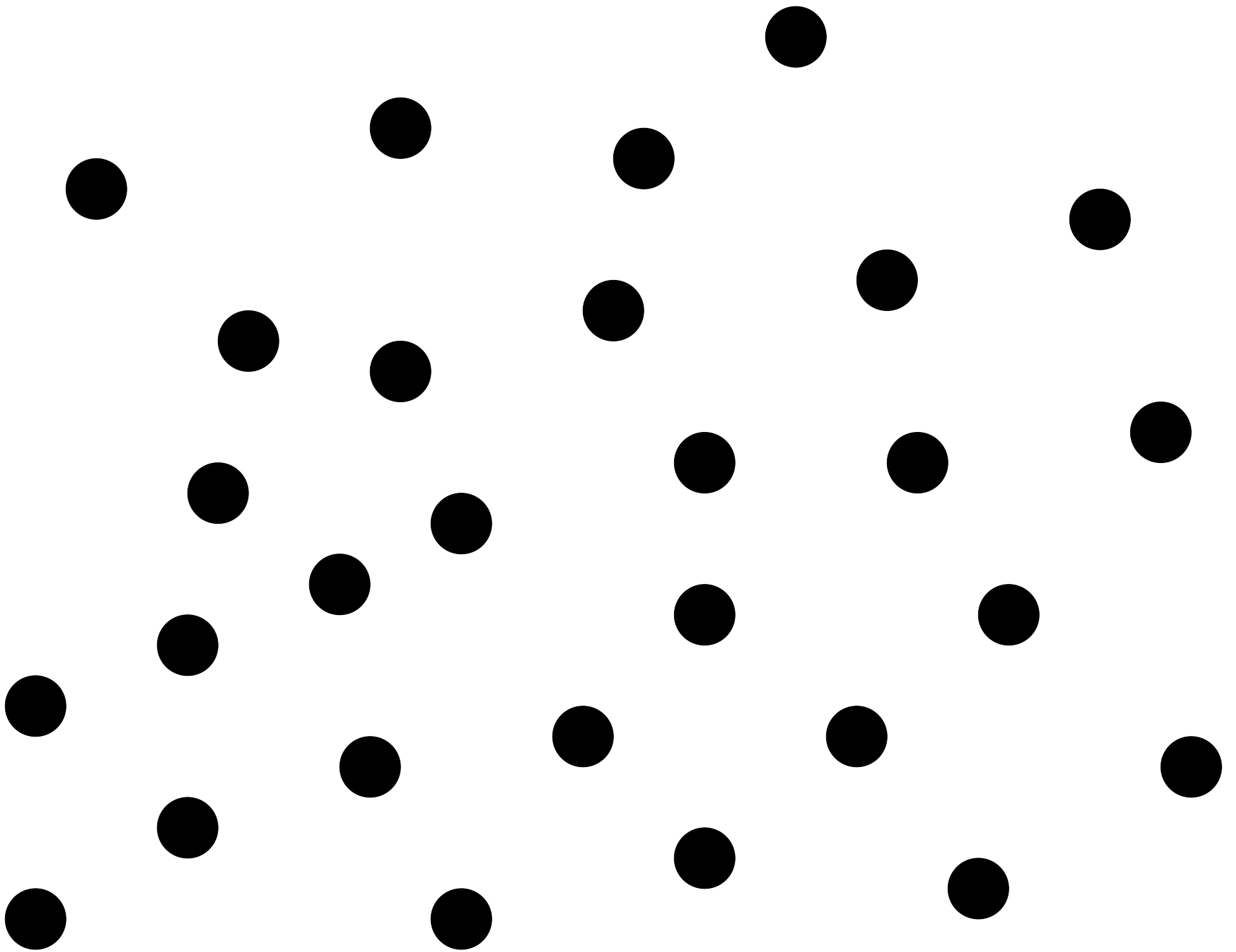
- Binary relations are an excellent way for capturing certain structures that appear in computer science.
- Today, we'll look at two examples (***partitions*** and ***prerequisites***).
- Then on Friday, we'll explore how to write proofs about definitions given in first-order logic.

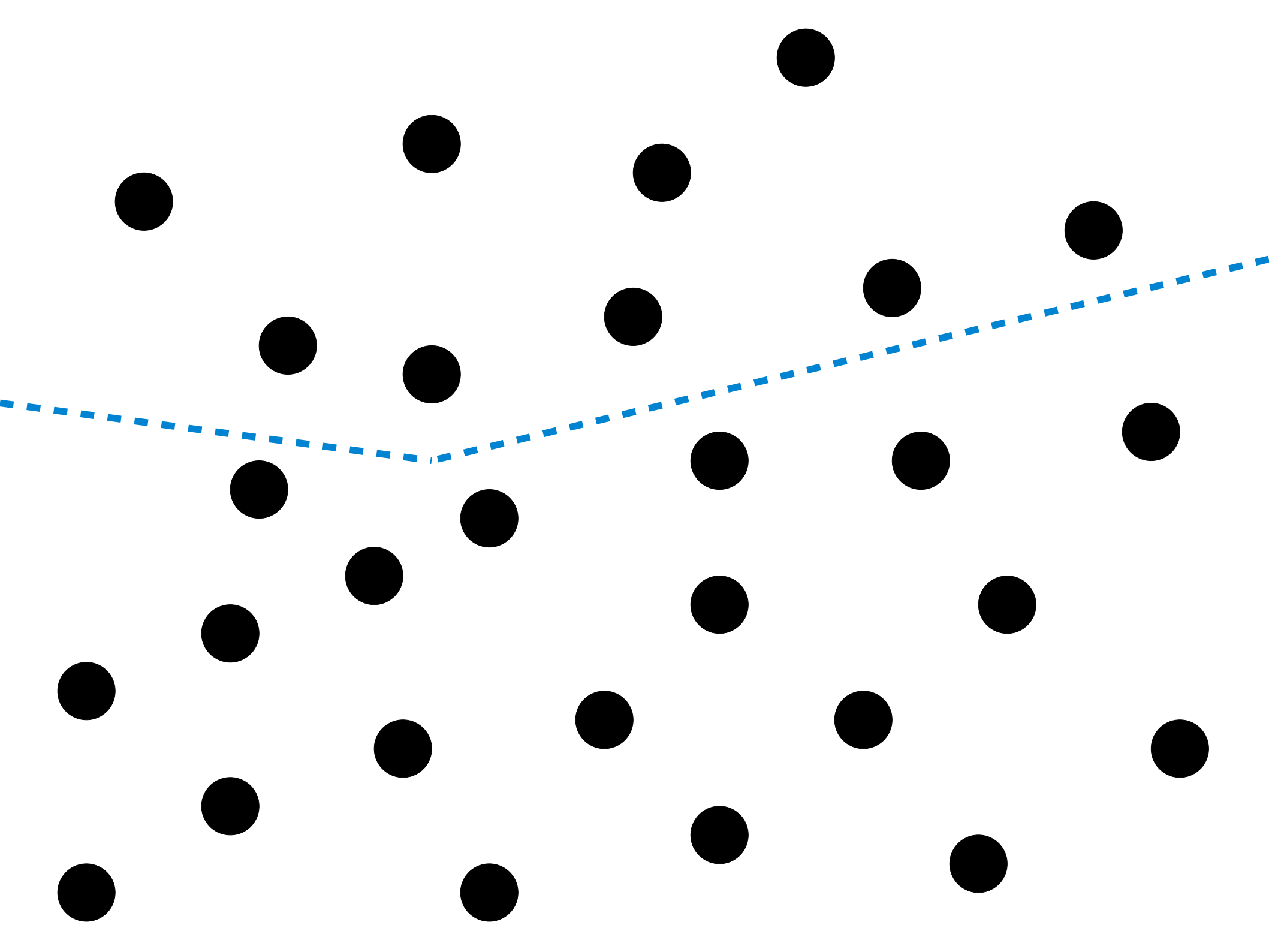
Partitions

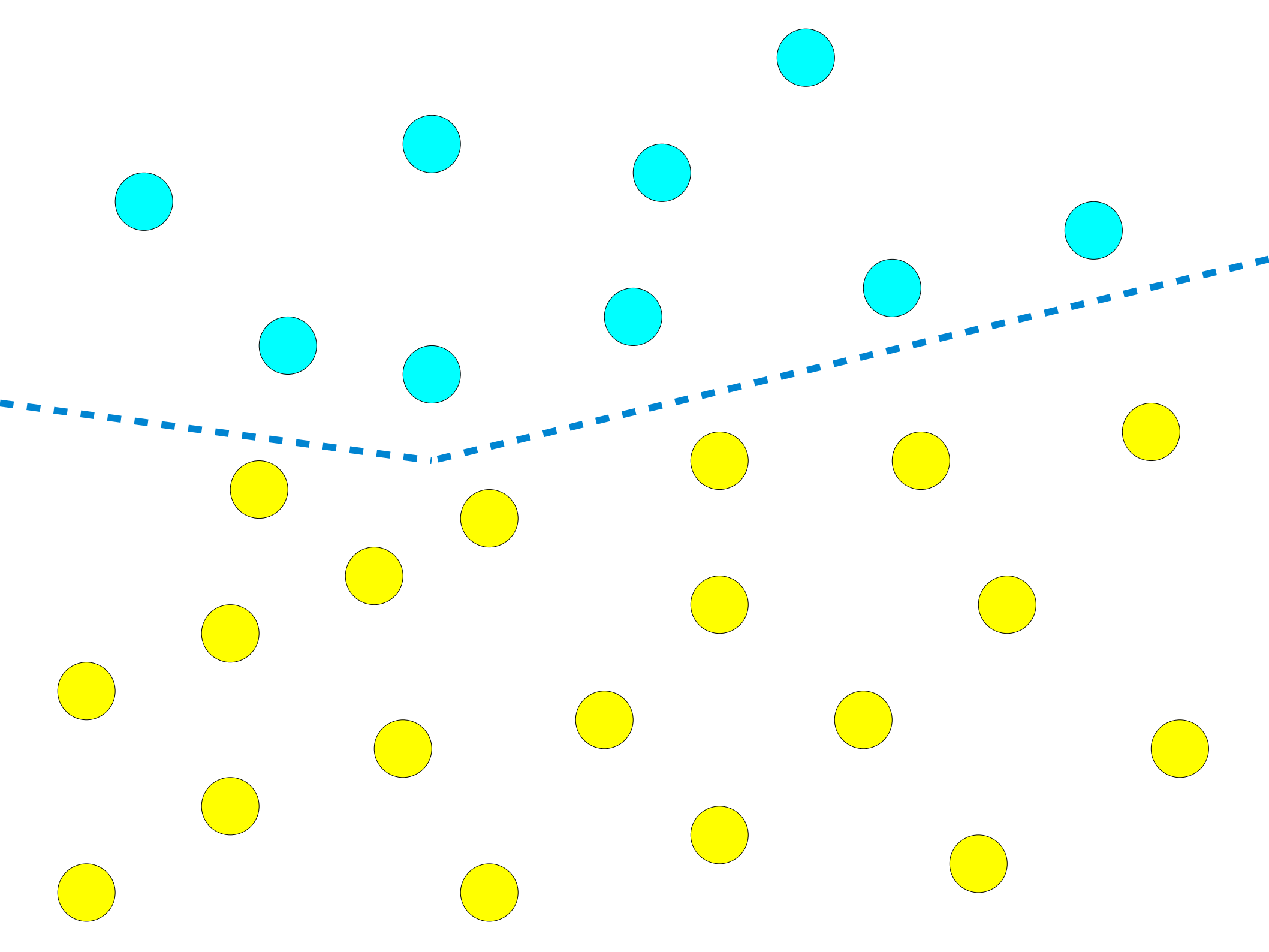


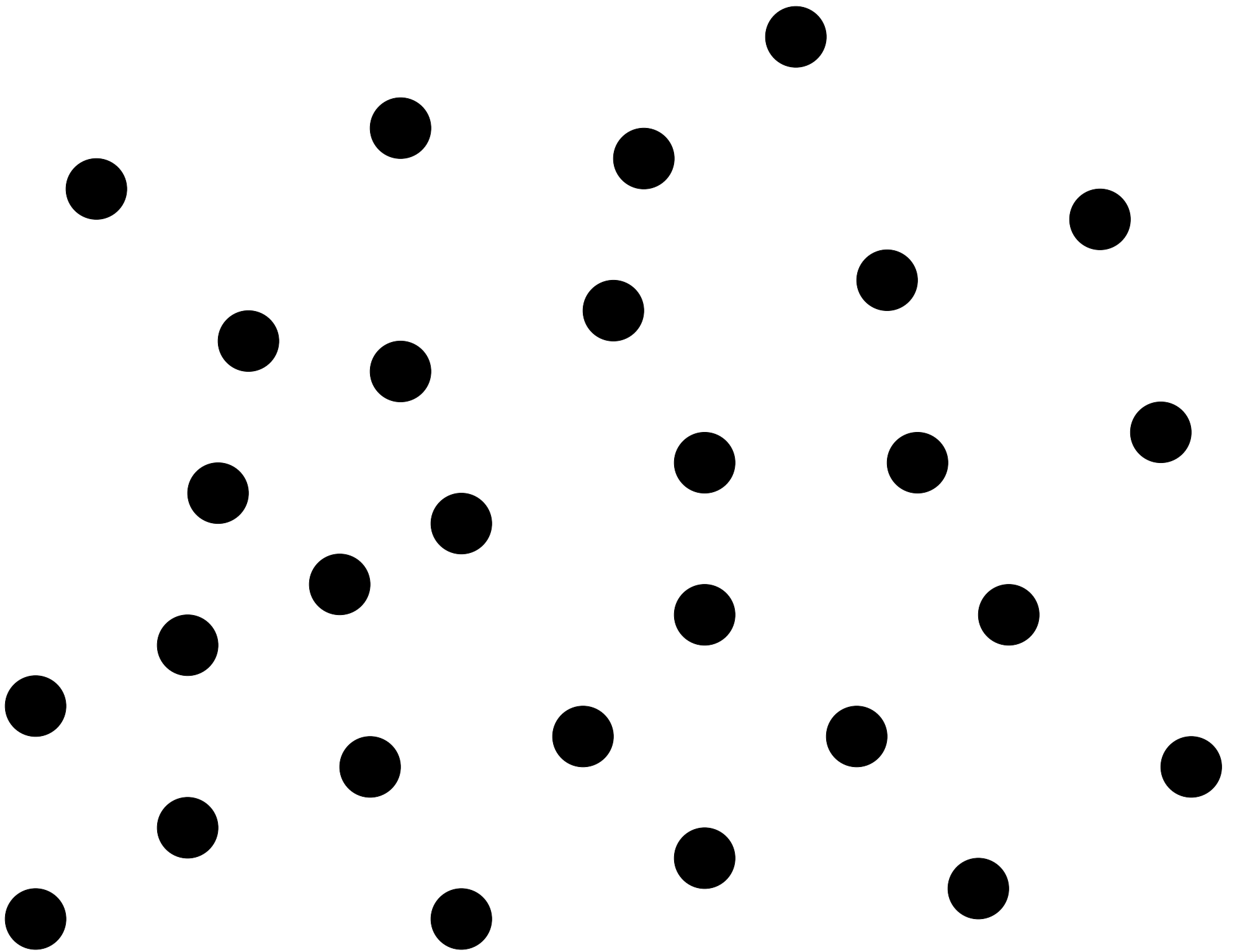


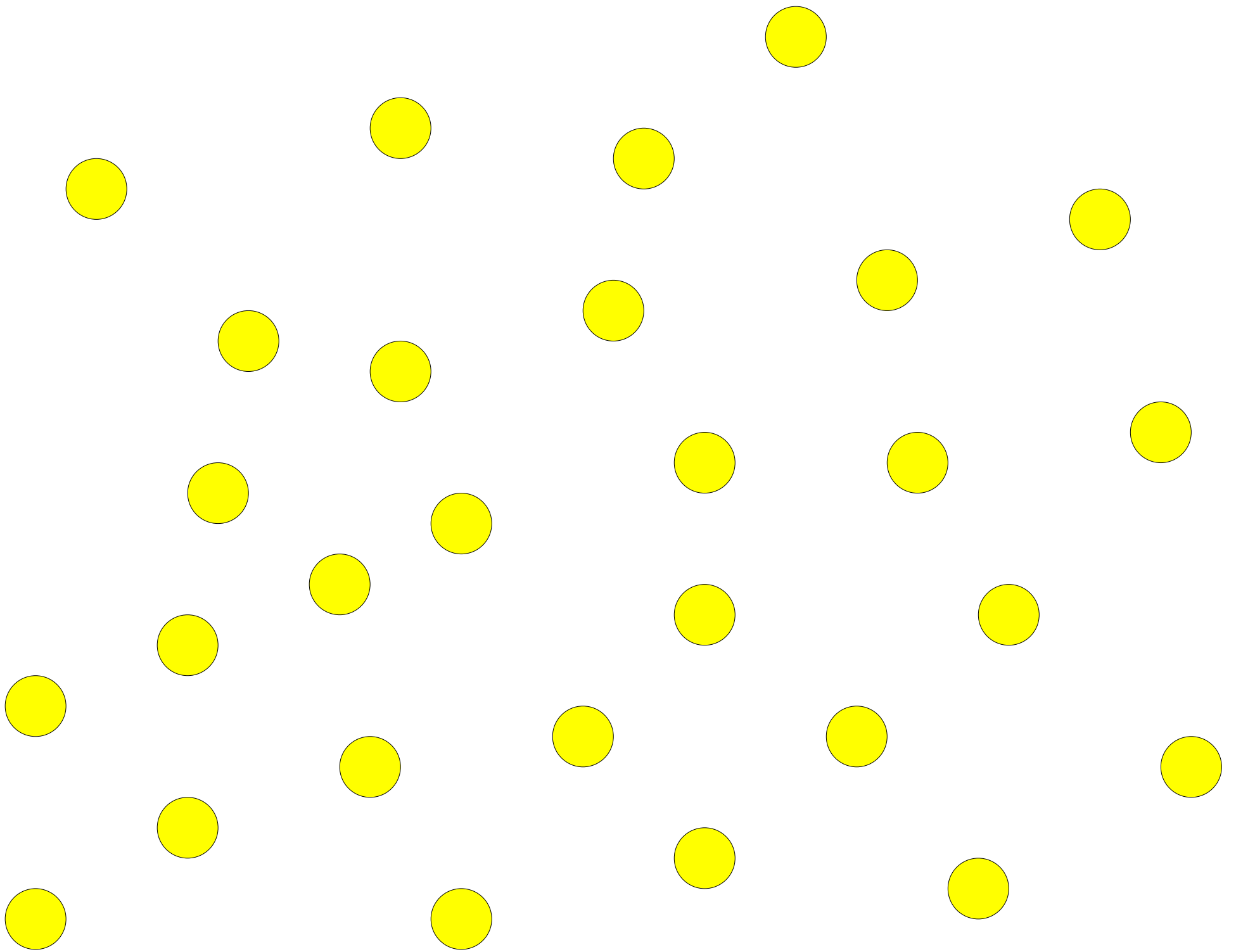


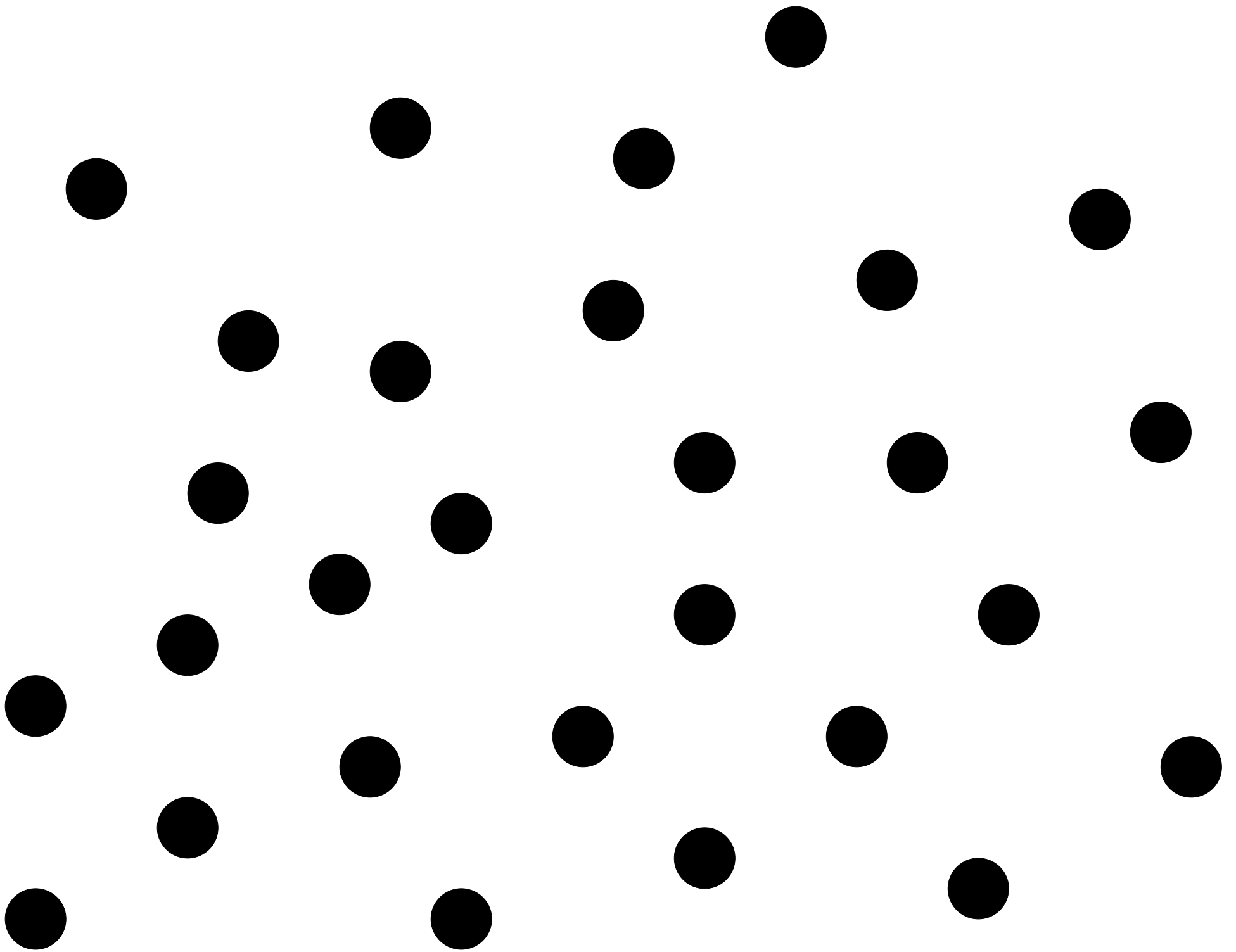


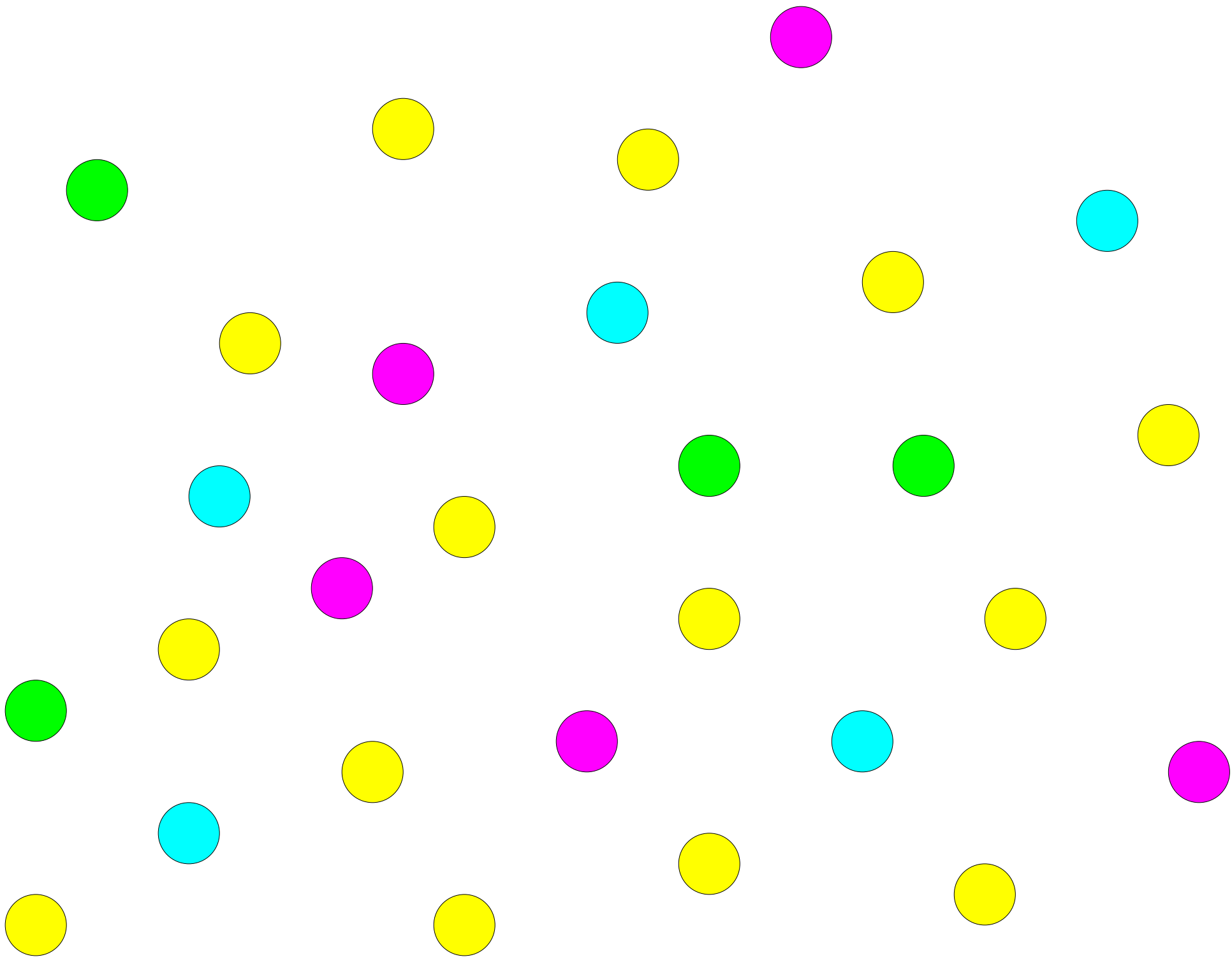












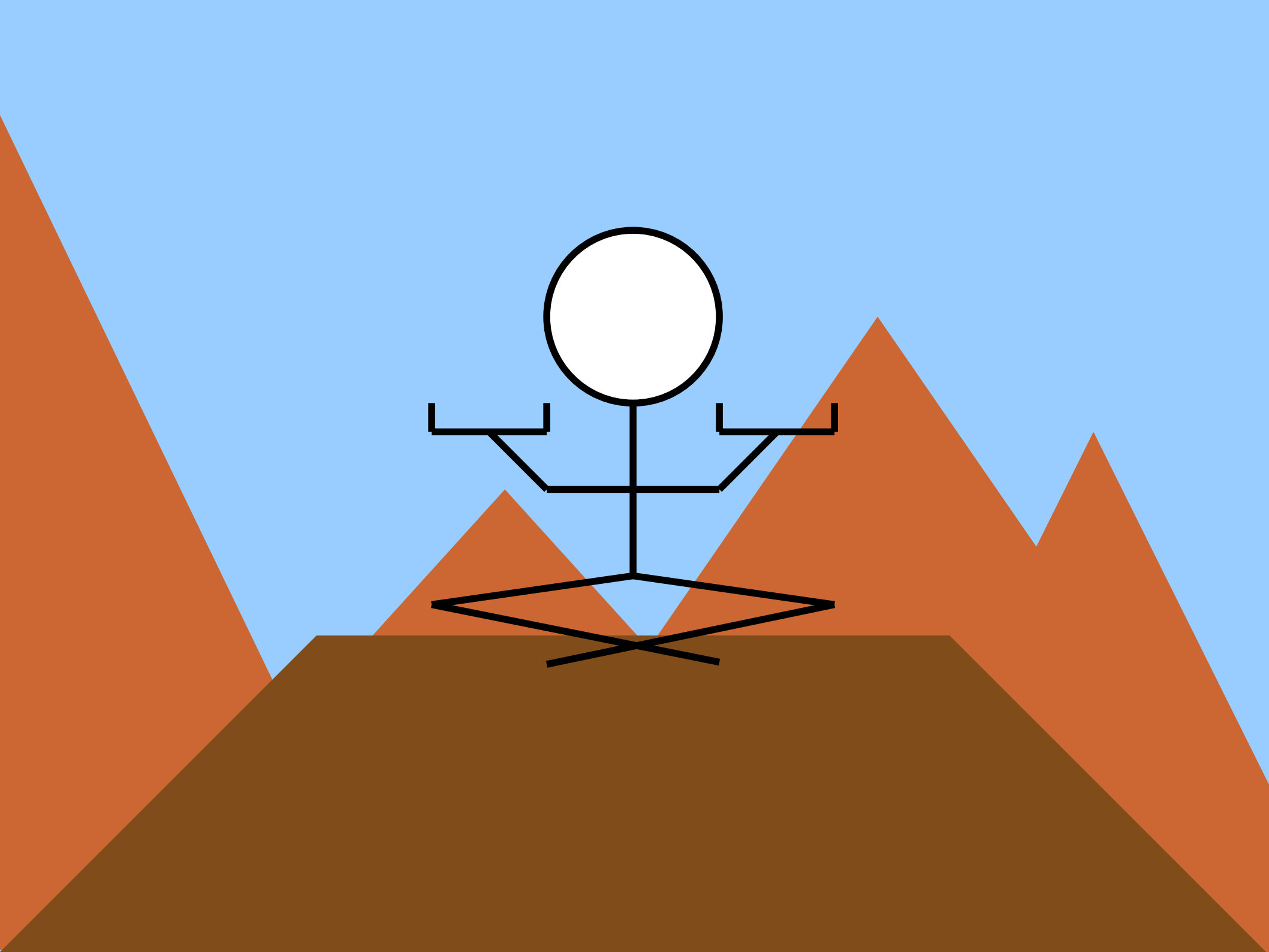
Partitions

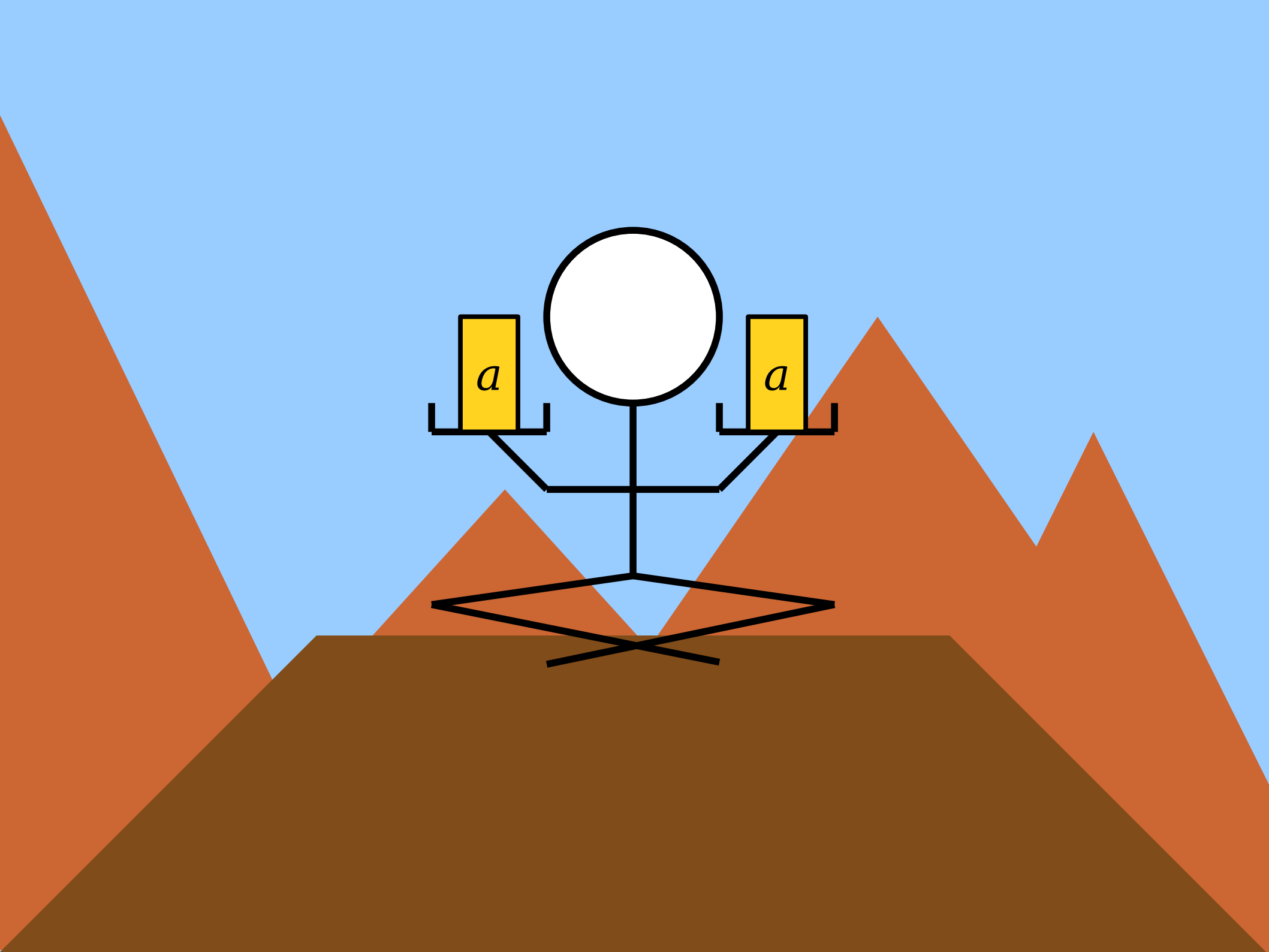
- A ***partition of a set*** is a way of splitting the set into disjoint, nonempty subsets so that every element belongs to exactly one subset.
- Two sets are ***disjoint*** if their intersection is the empty set; formally, sets S and T are disjoint if $S \cap T = \emptyset$.
- Intuitively, a partition of a set breaks the set apart into smaller pieces.
- There doesn't have to be any rhyme or reason to what those pieces are, though often there is one.

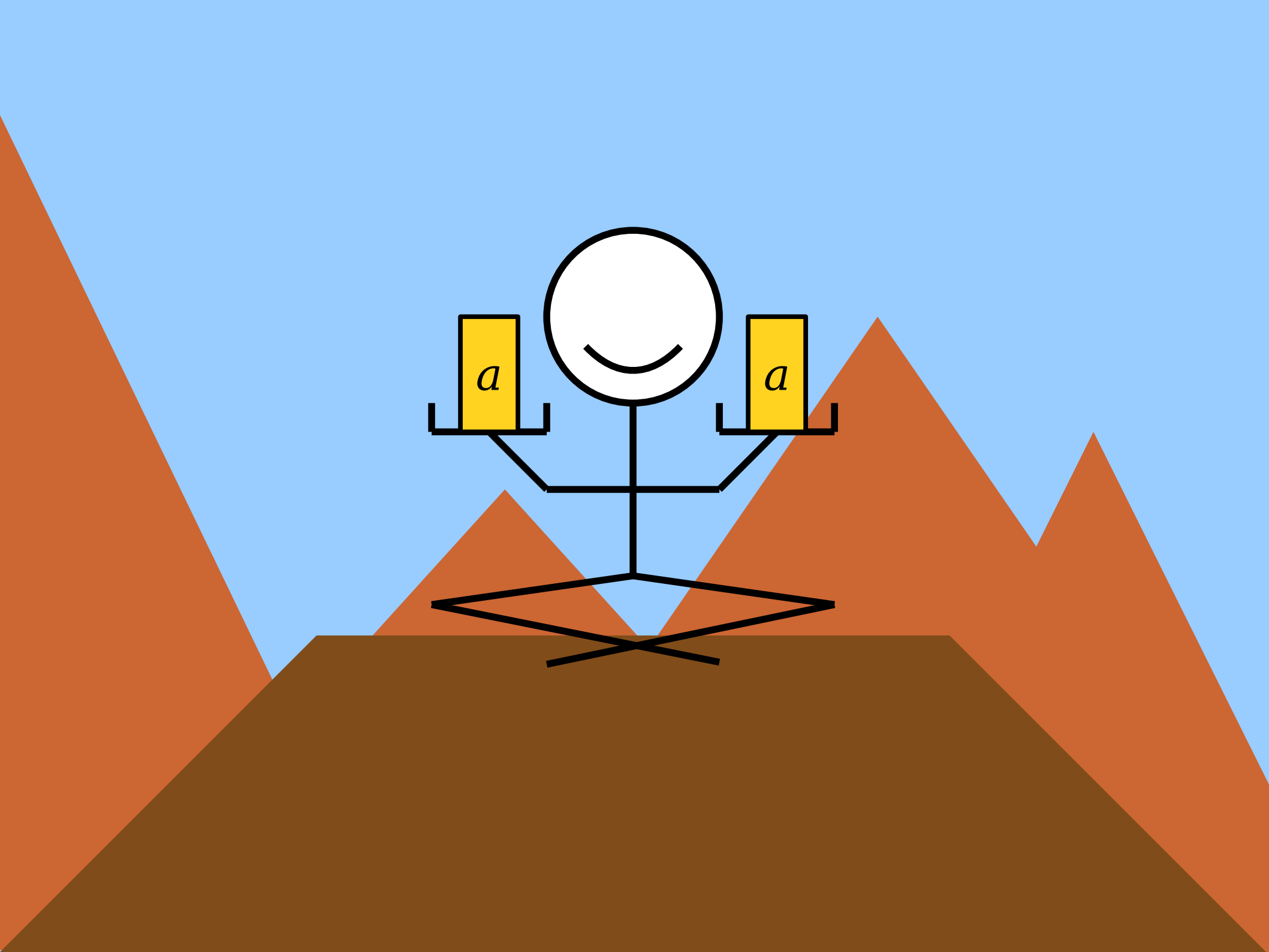
Partitions and Clustering

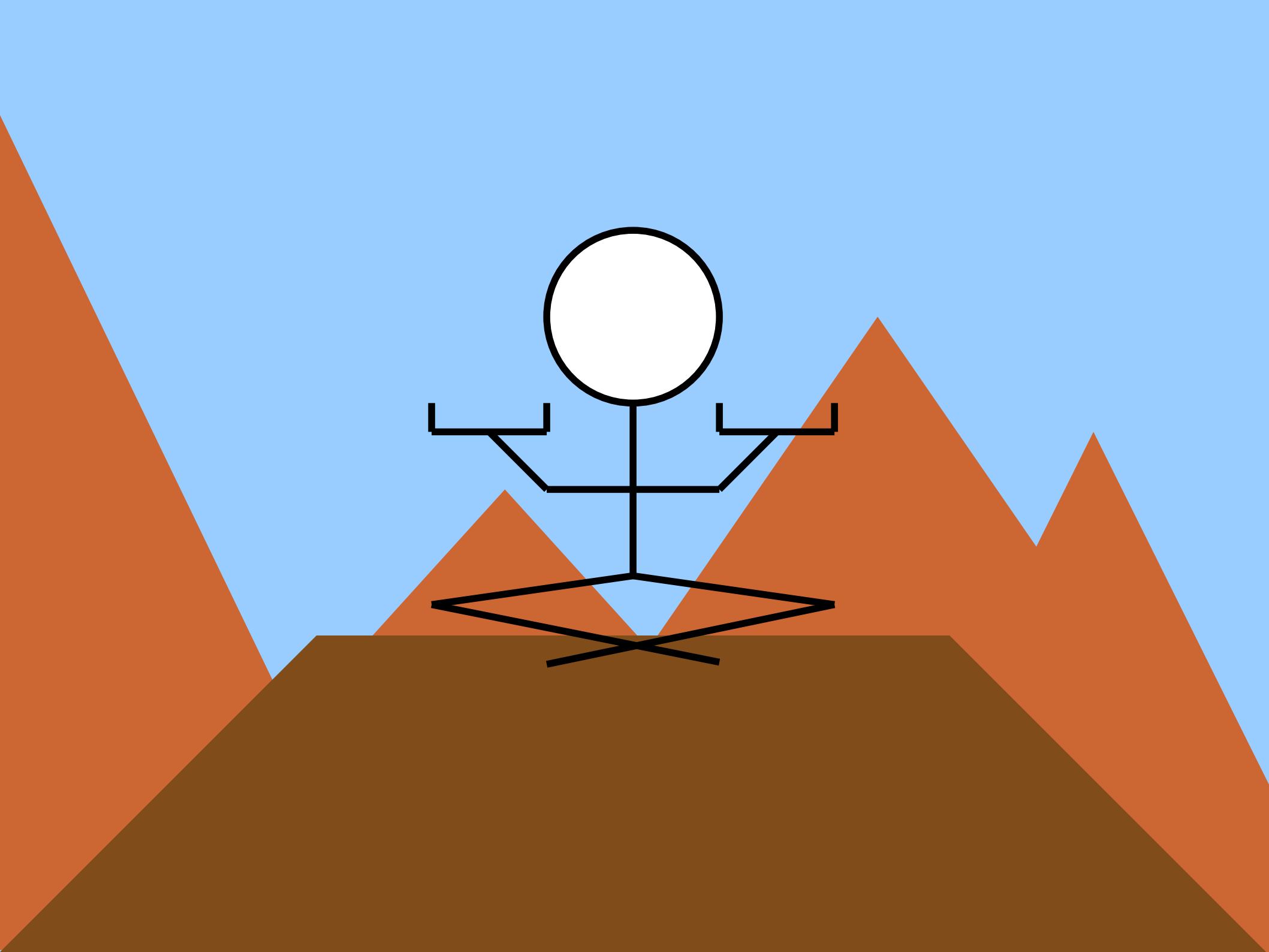
- If you have a set of data, you can often learn something from the data by finding a “good” partition of that data and inspecting the partitions.
- Usually, the term ***clustering*** is used in data analysis rather than *partitioning*.
- Interested to learn more? Take CS161 or CS246!

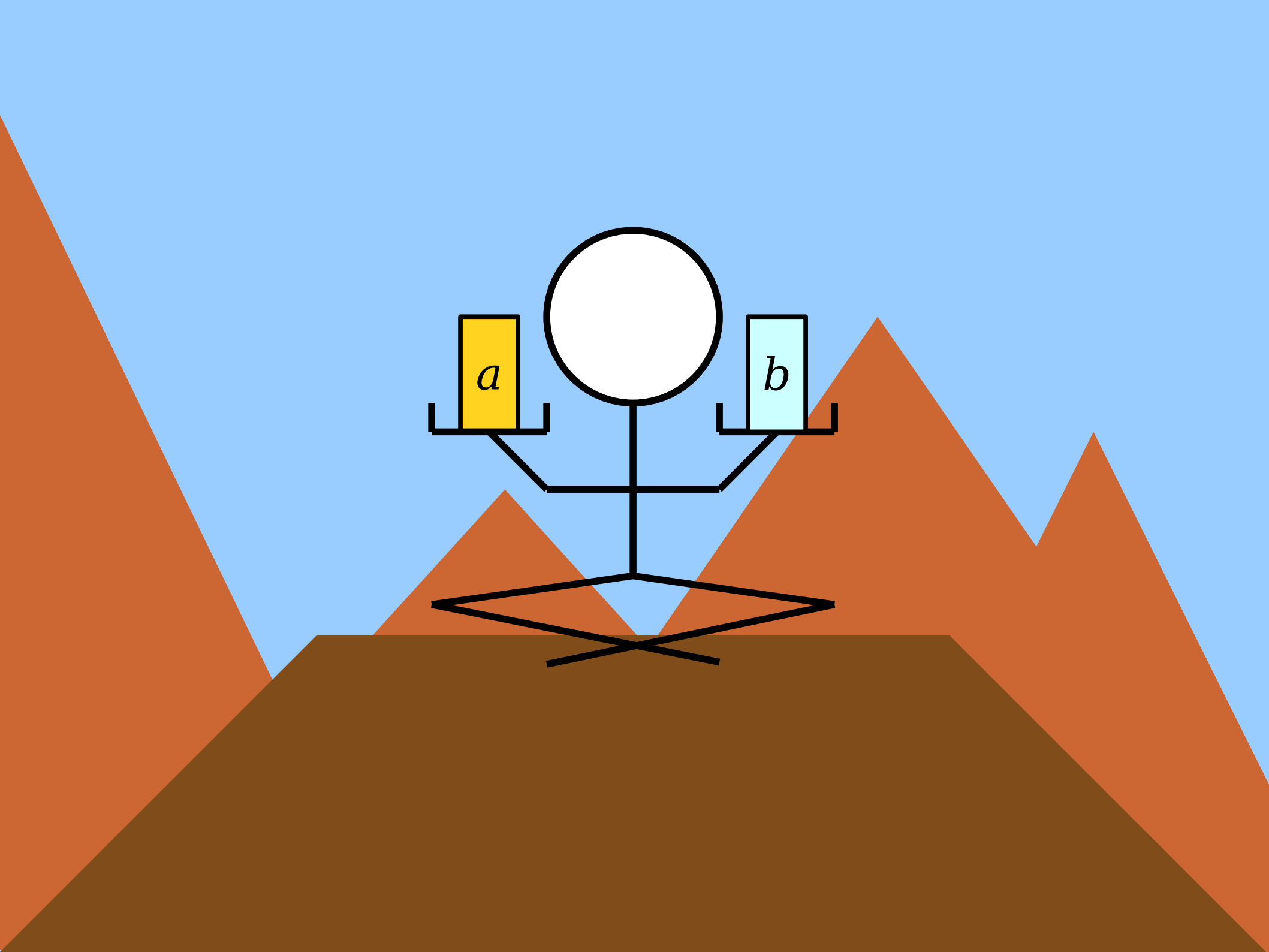
What's the connection between partitions
and binary relations?

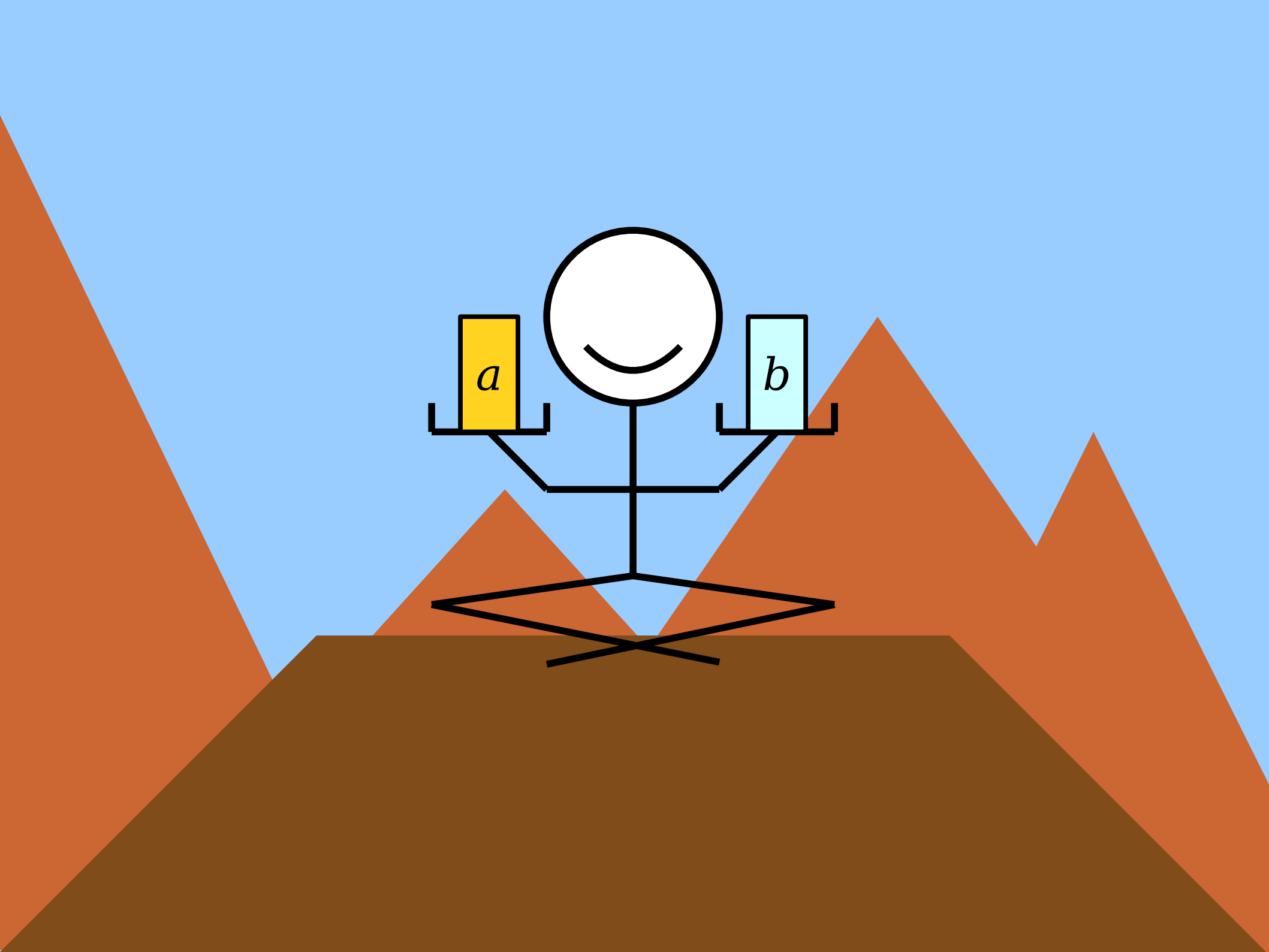


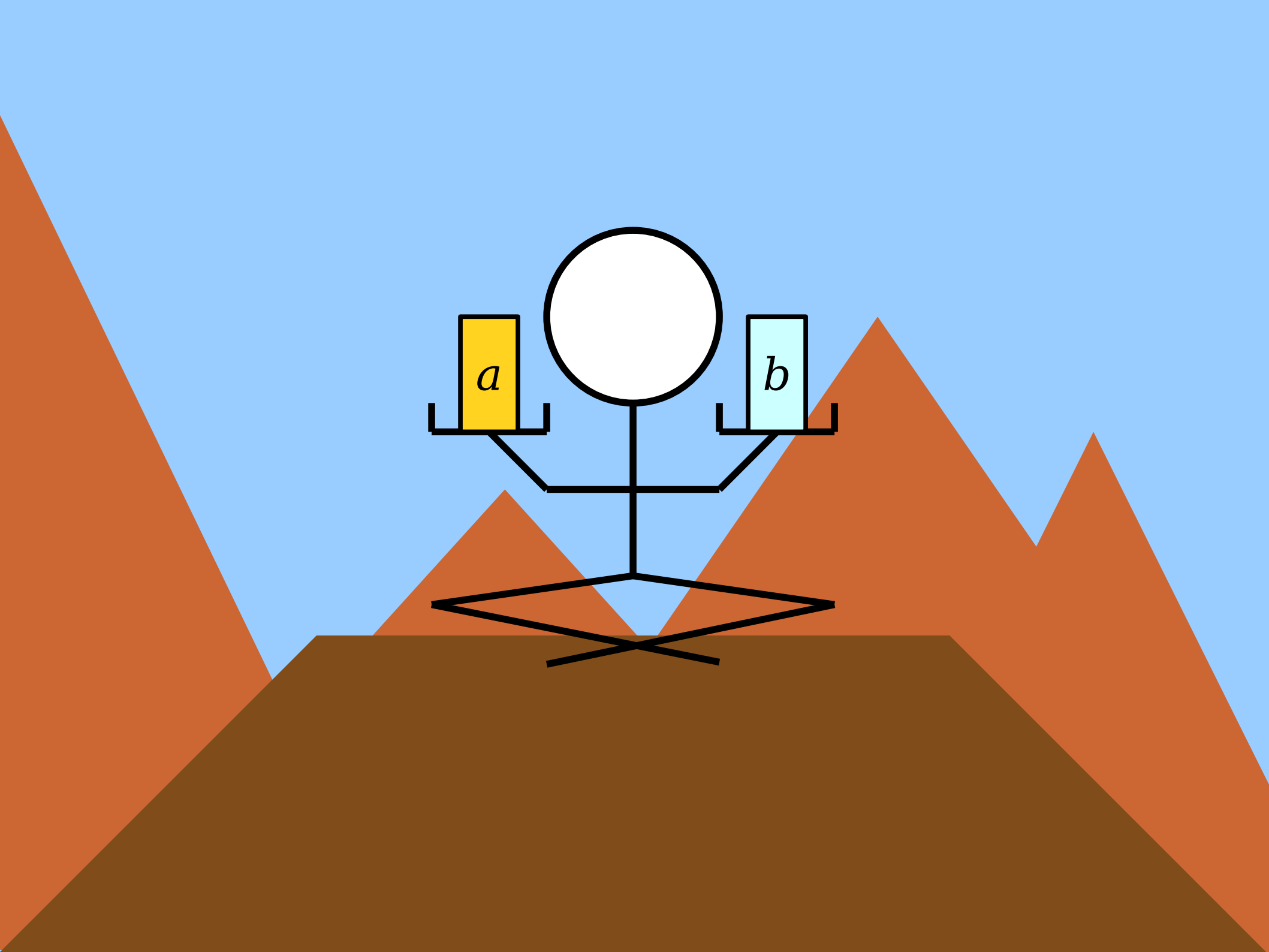




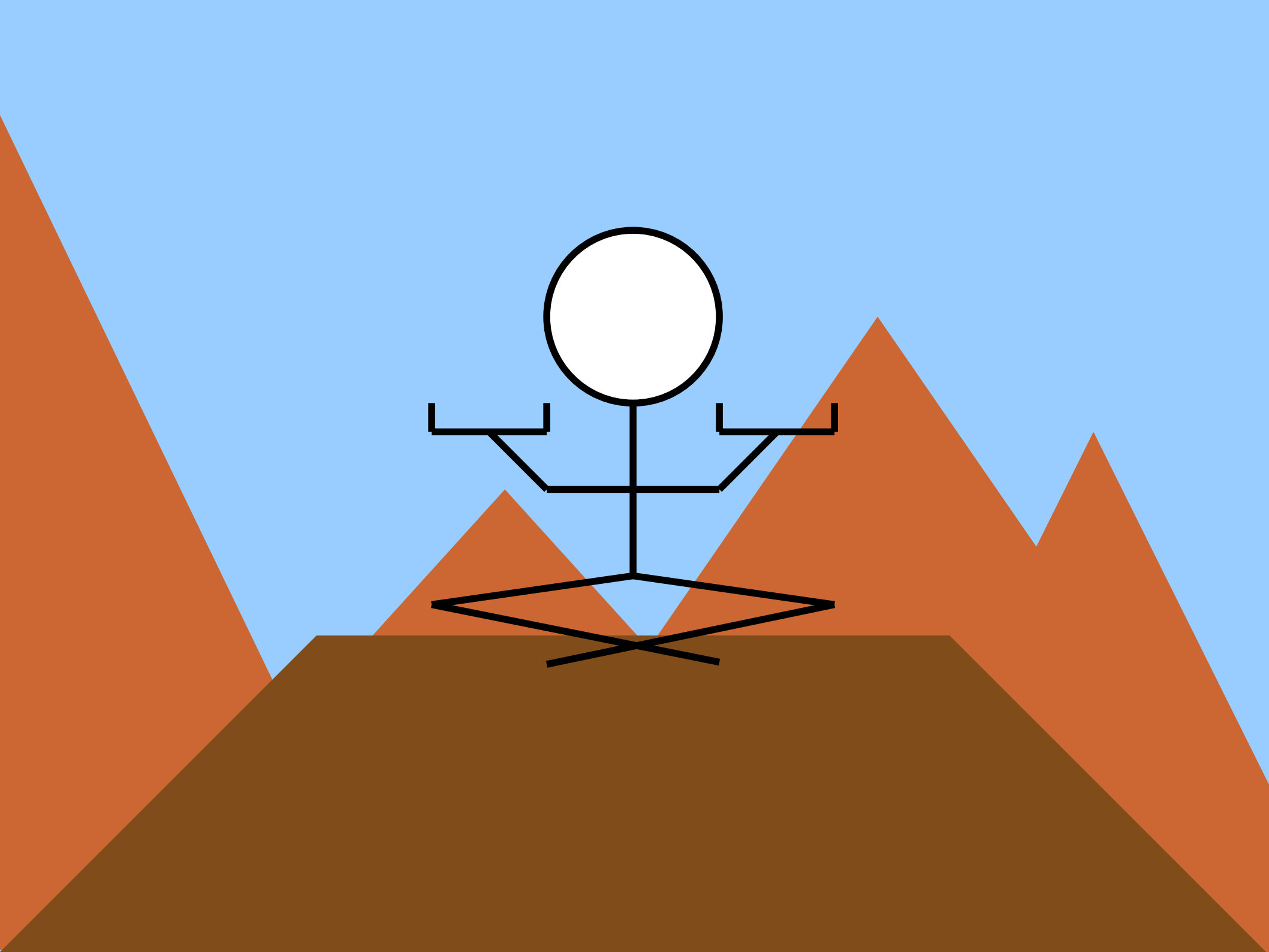


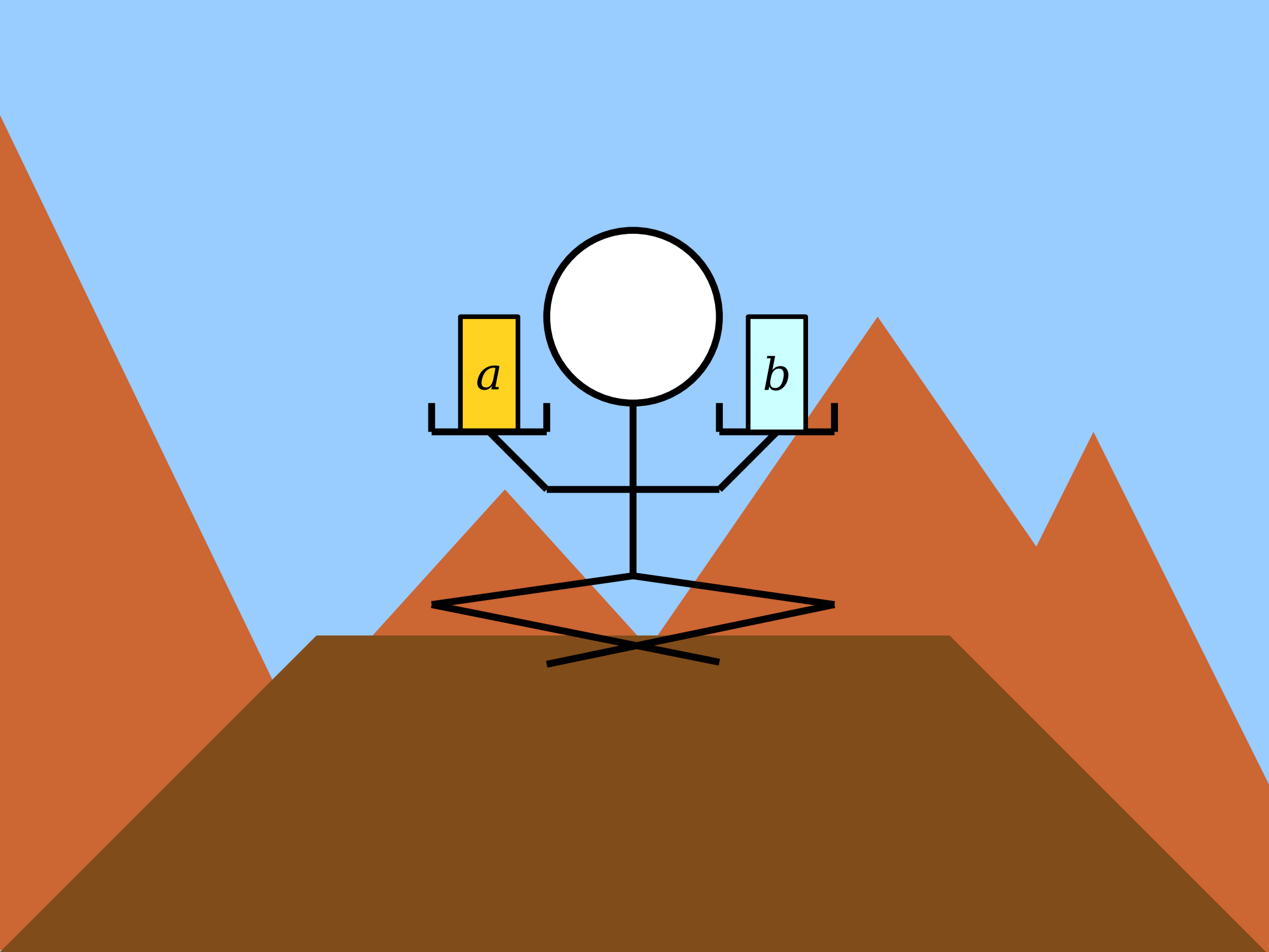


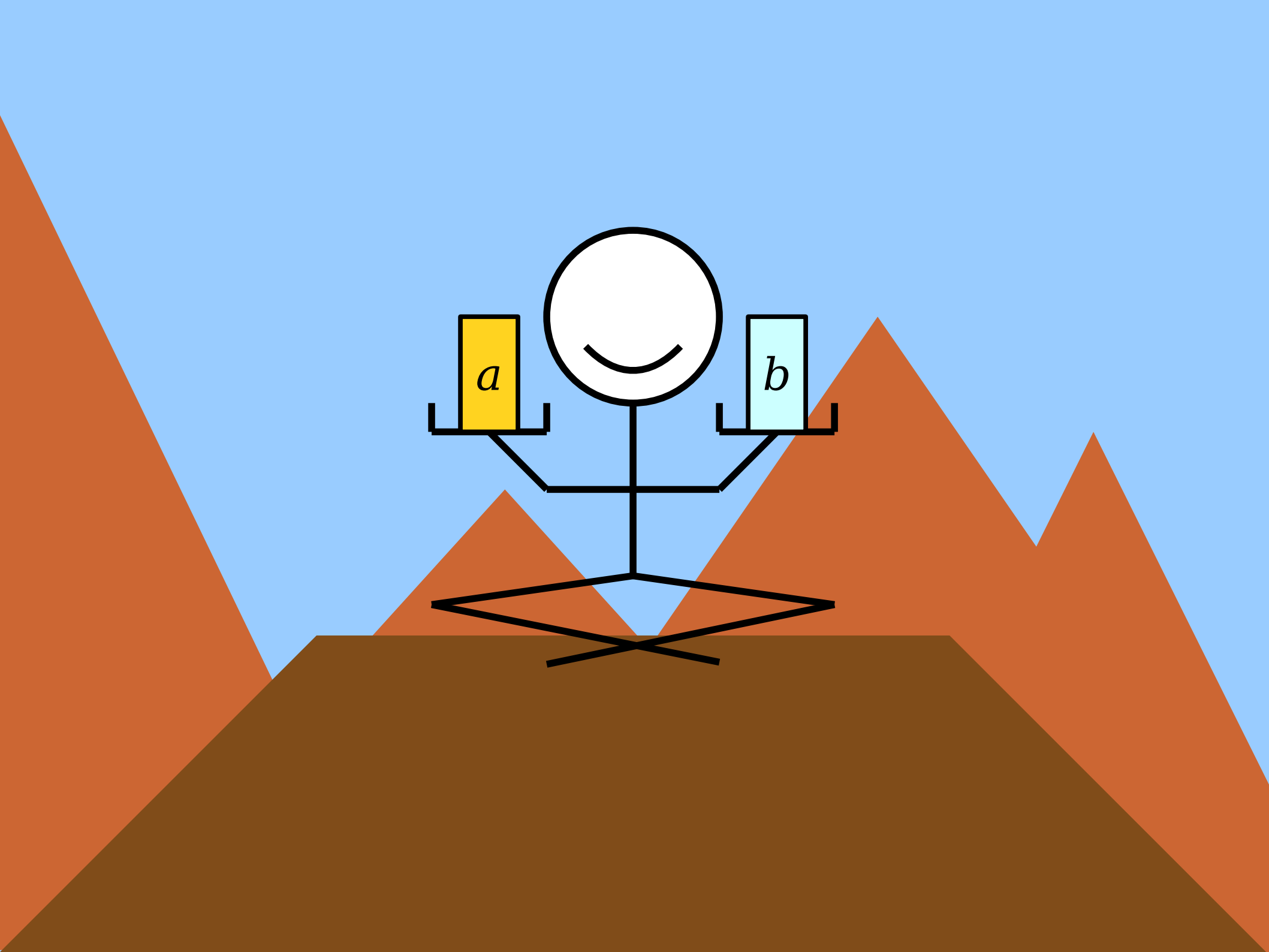


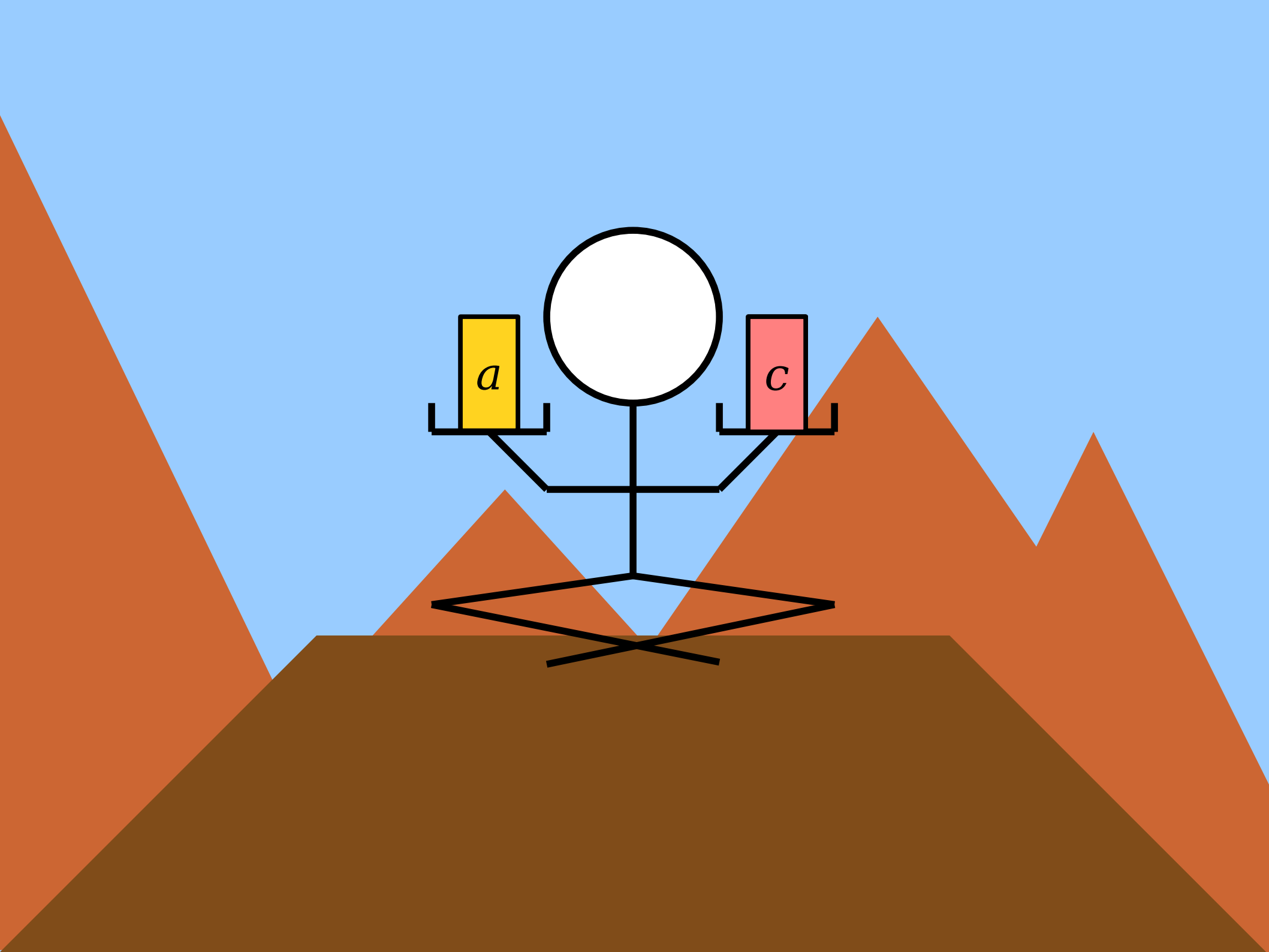




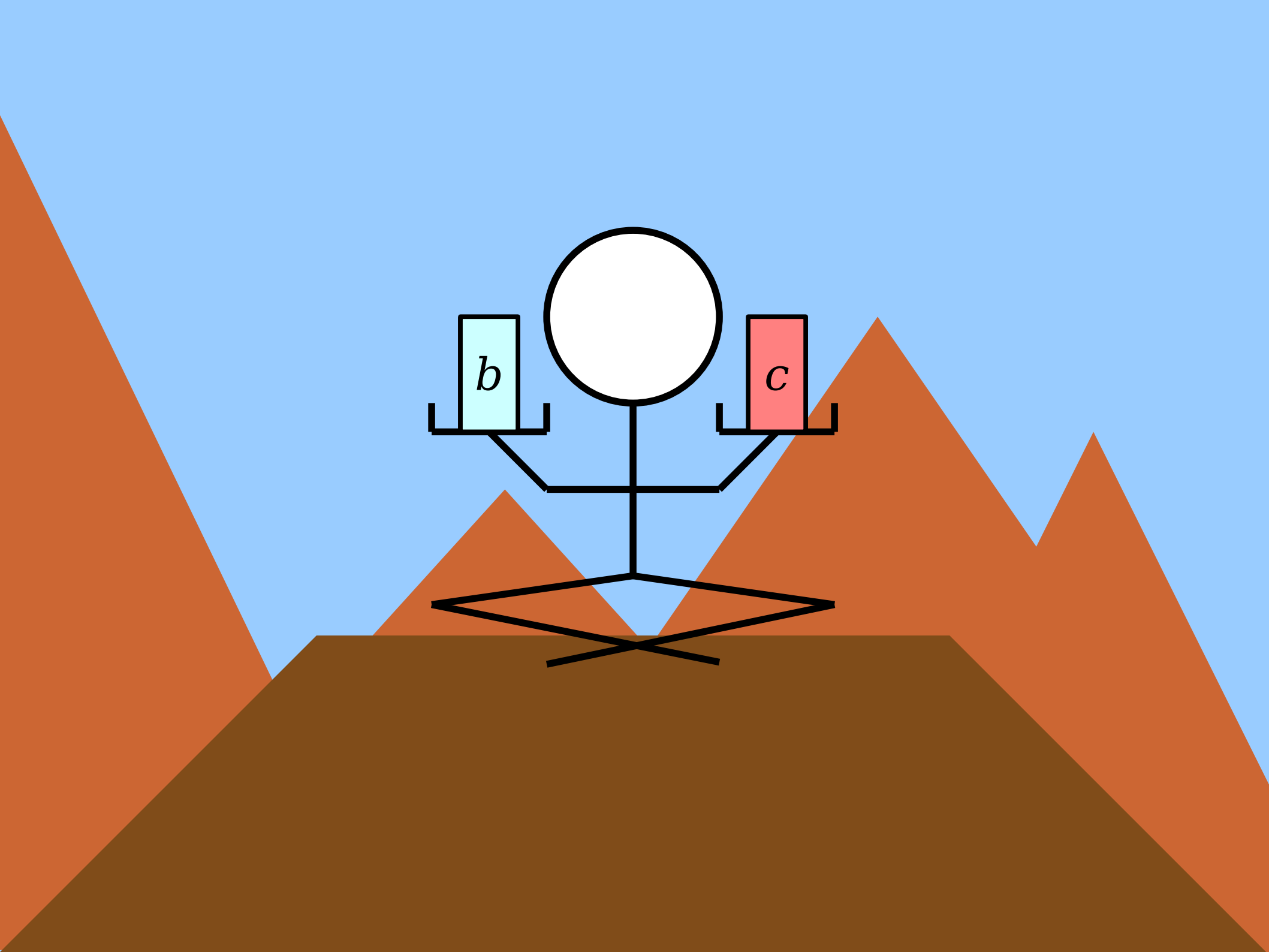


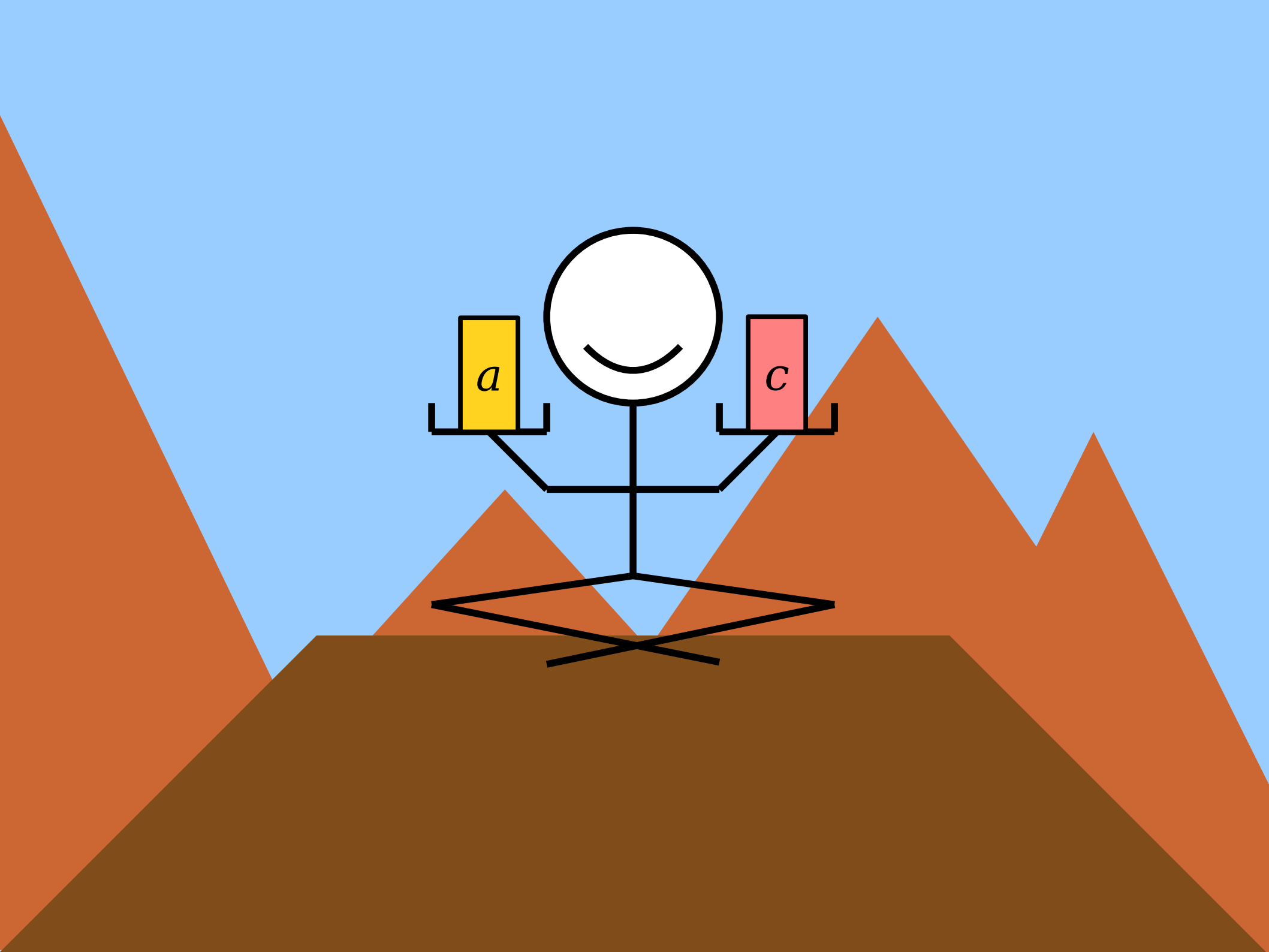






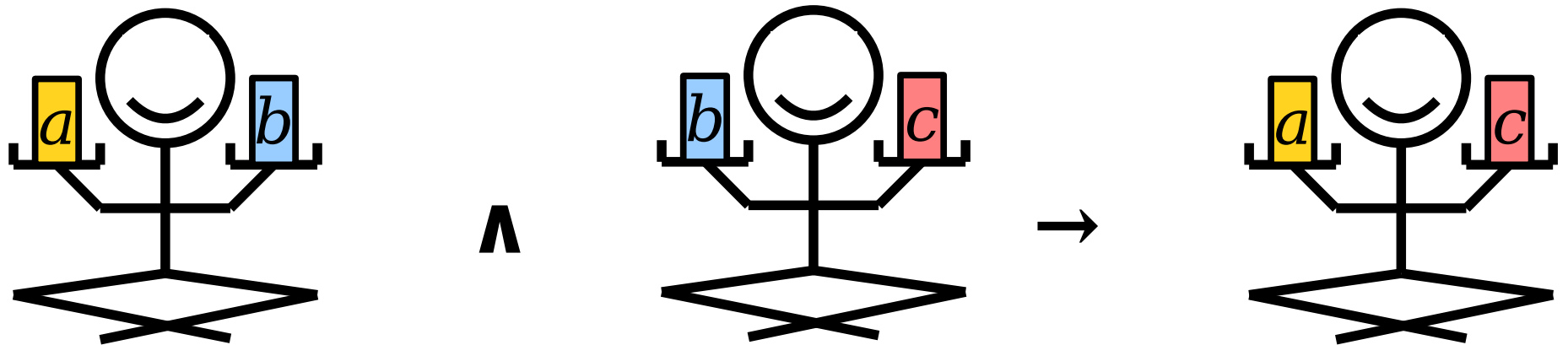
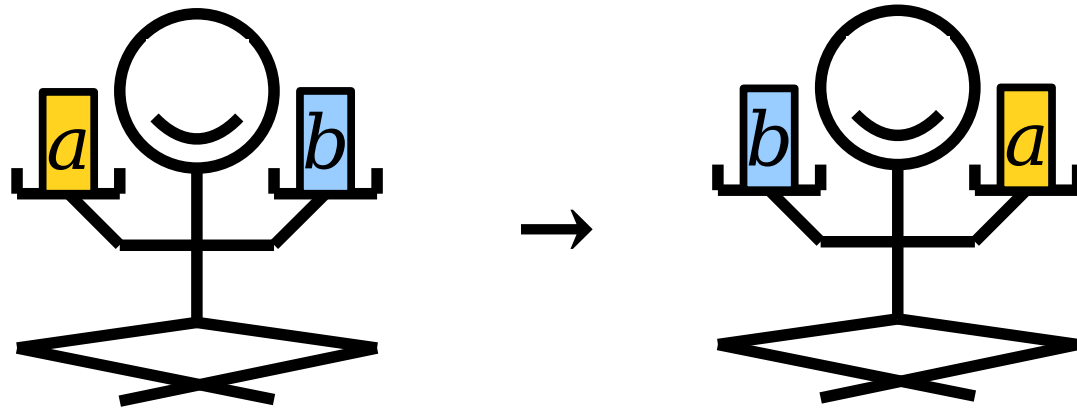
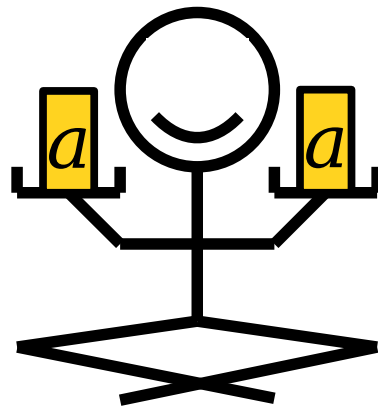






a

c



aRa

$aRb \rightarrow bRa$

$aRb \wedge bRc \rightarrow aRc$

$$\forall a \in A. aRa$$

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

$$\forall a \in A. aRa$$

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

Reflexivity

Some relations always hold from any element to itself.

Examples:

- $x = x$ for any x .
- $A \subseteq A$ for any set A .
- $x \equiv_k x$ for any x .

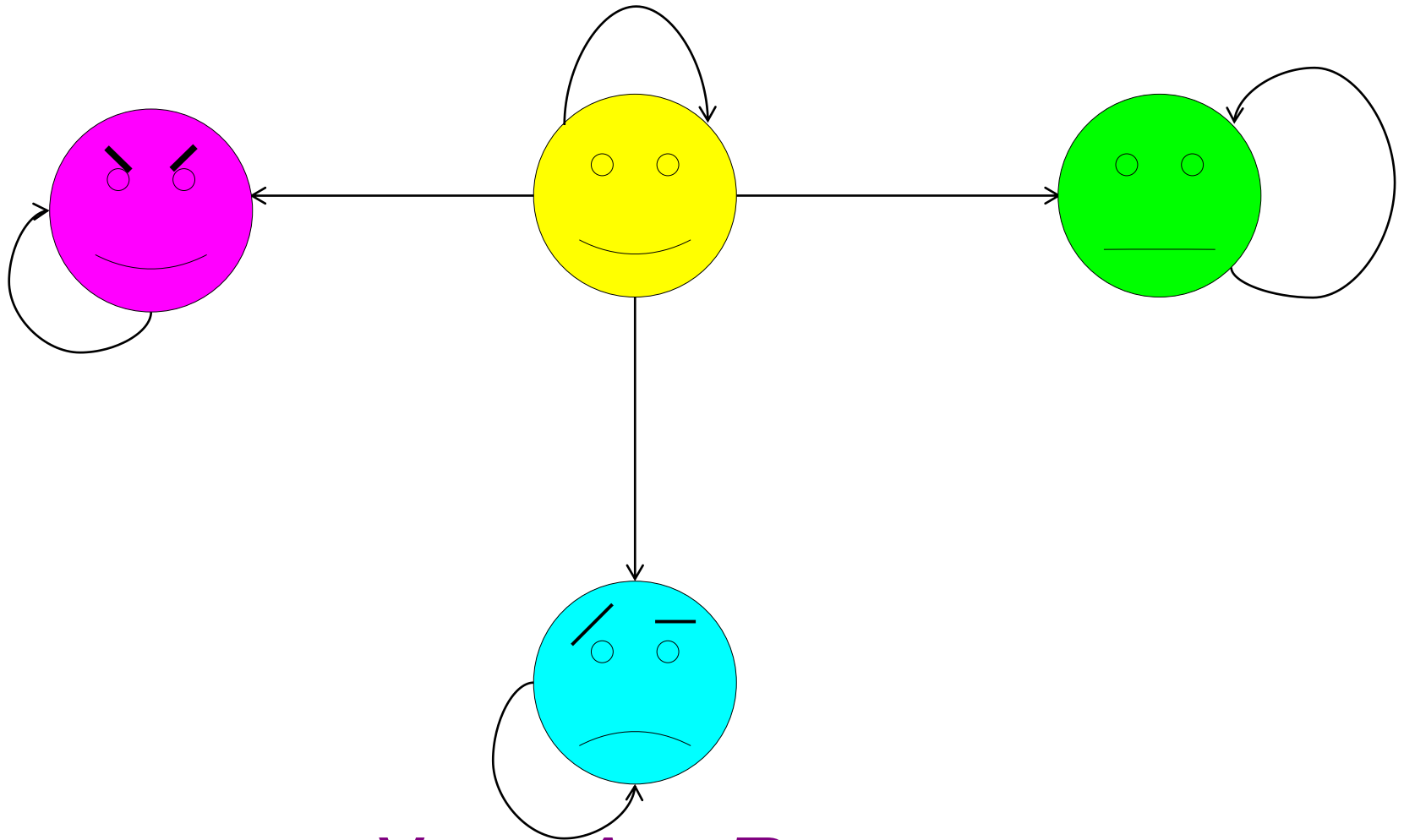
Relations of this sort are called ***reflexive***.

Formally speaking, a binary relation R over a set A is reflexive if the following first-order statement is true:

$$\forall a \in A. aRa$$

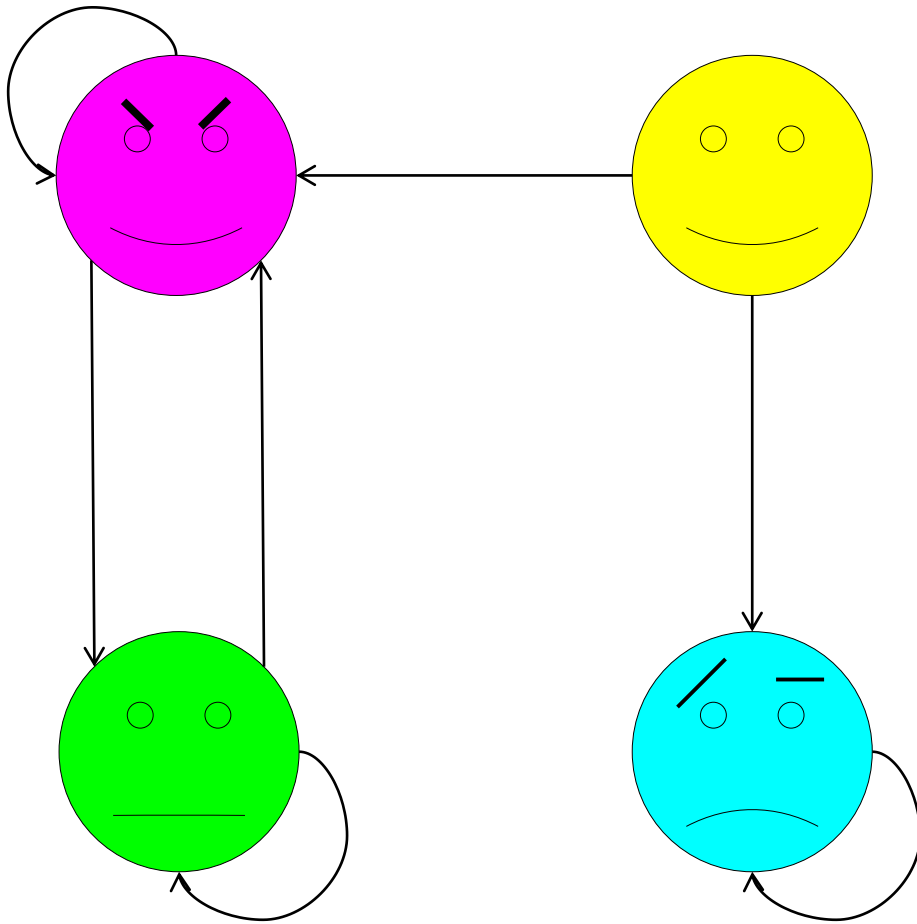
(“*Every element is related to itself.*”)

Reflexivity Visualized



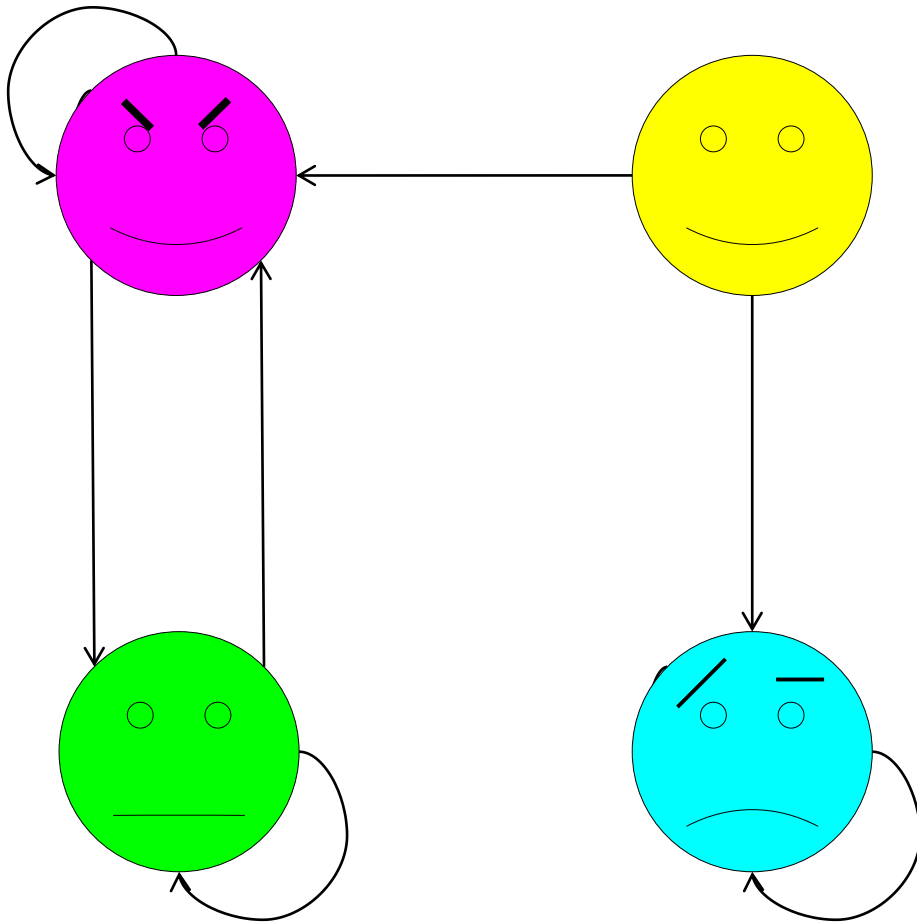
$\forall a \in A. aRa$

(“Every element is related to itself.”)

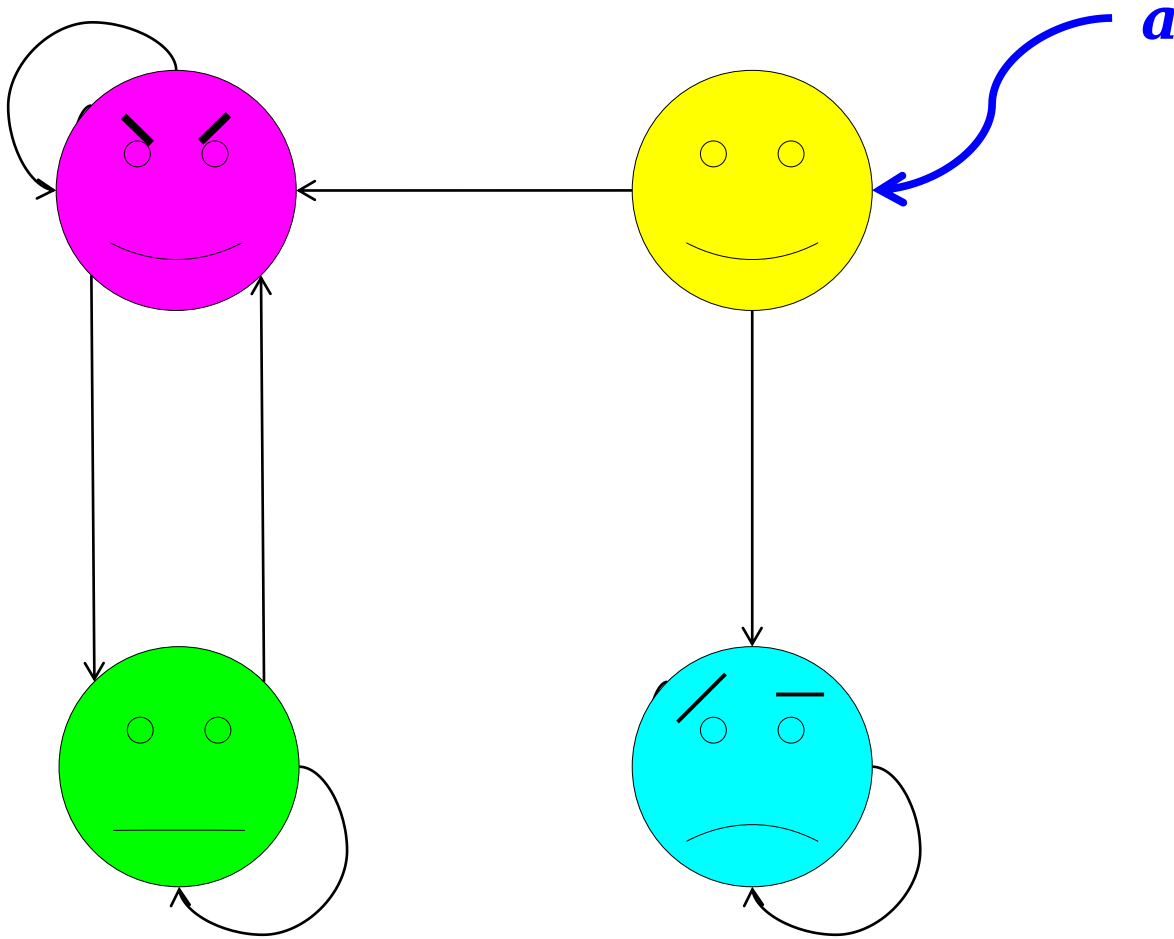


Let R be the relation drawn to the left. Is R reflexive?

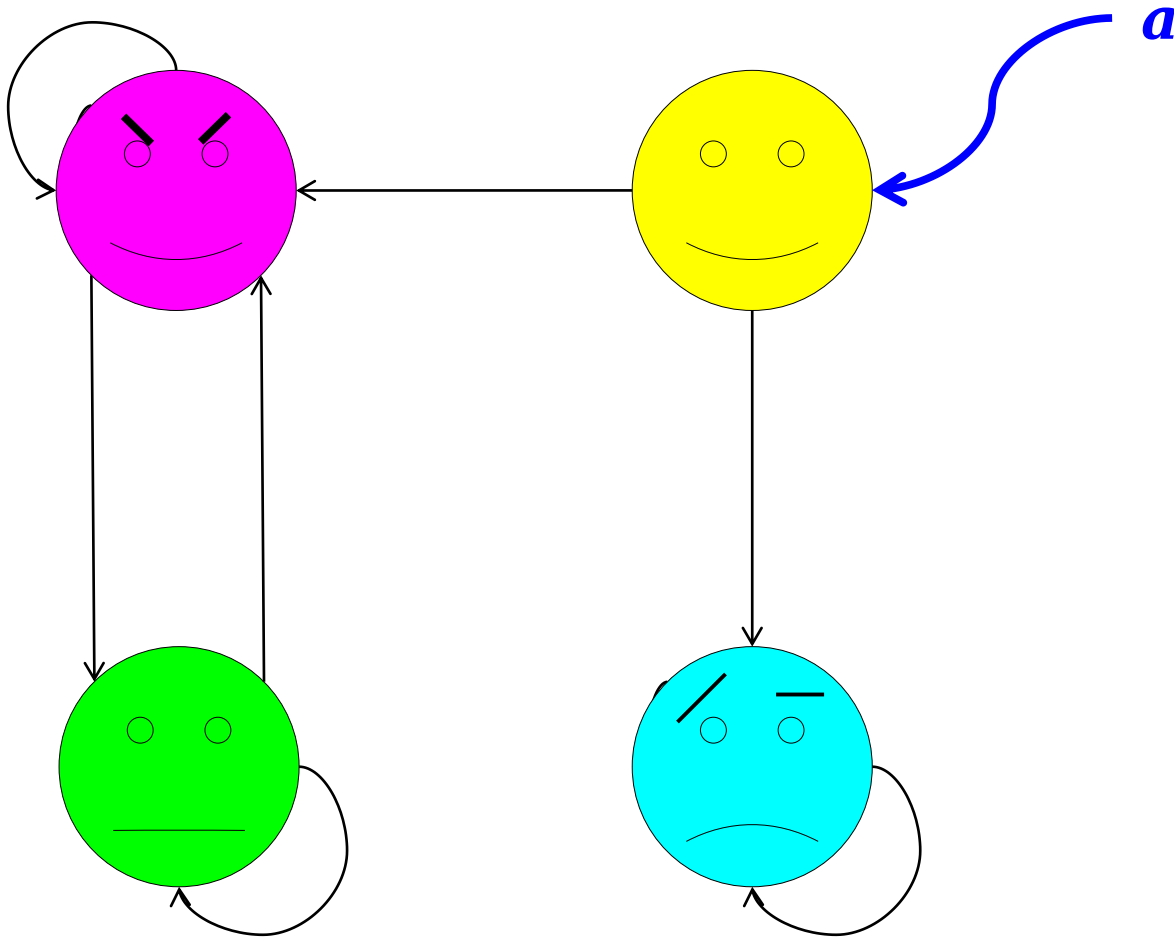
$\forall a \in A. aRa$
(“Every element is related to itself.”)



$\forall a \in A. aRa$
(*“Every element is related to itself.”*)

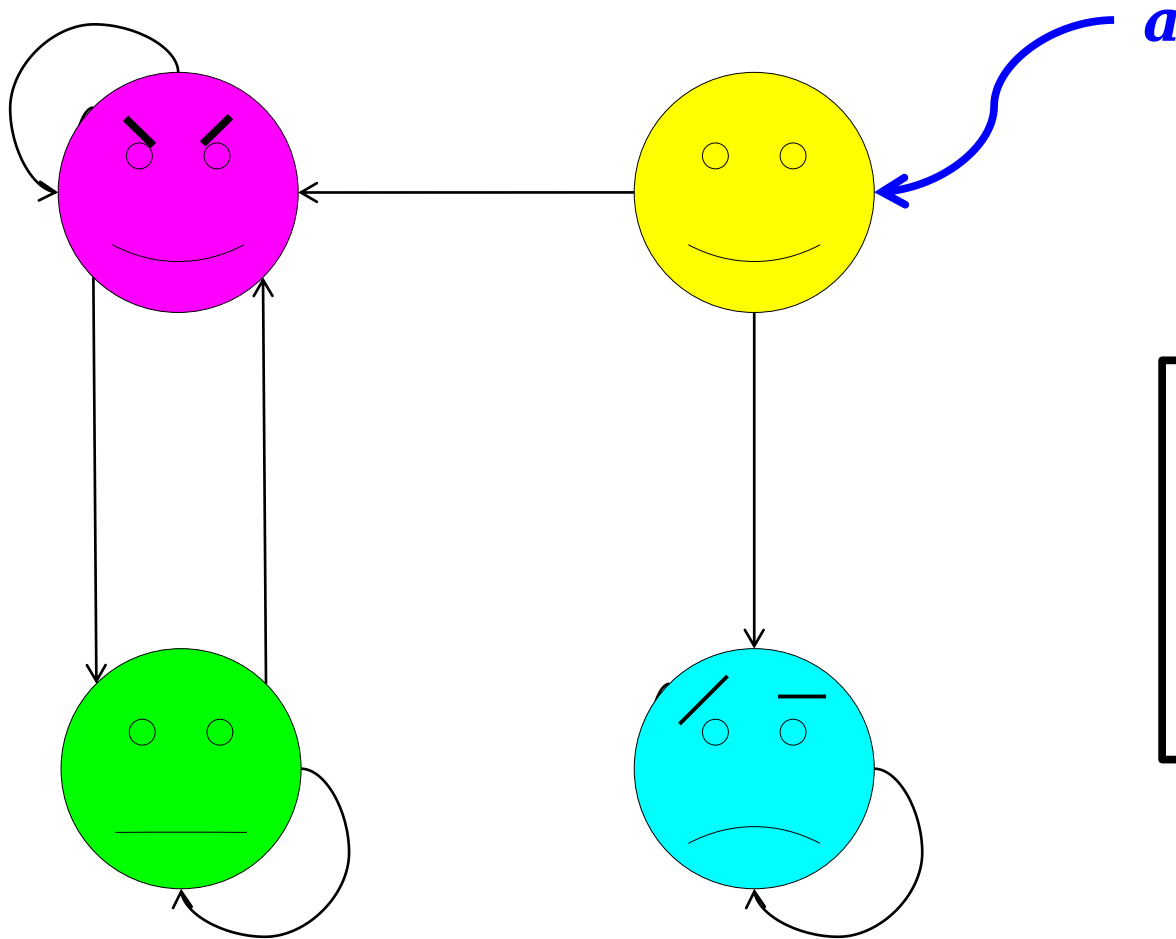


$\forall a \in A. aRa$
(*“Every element is related to itself.”*)



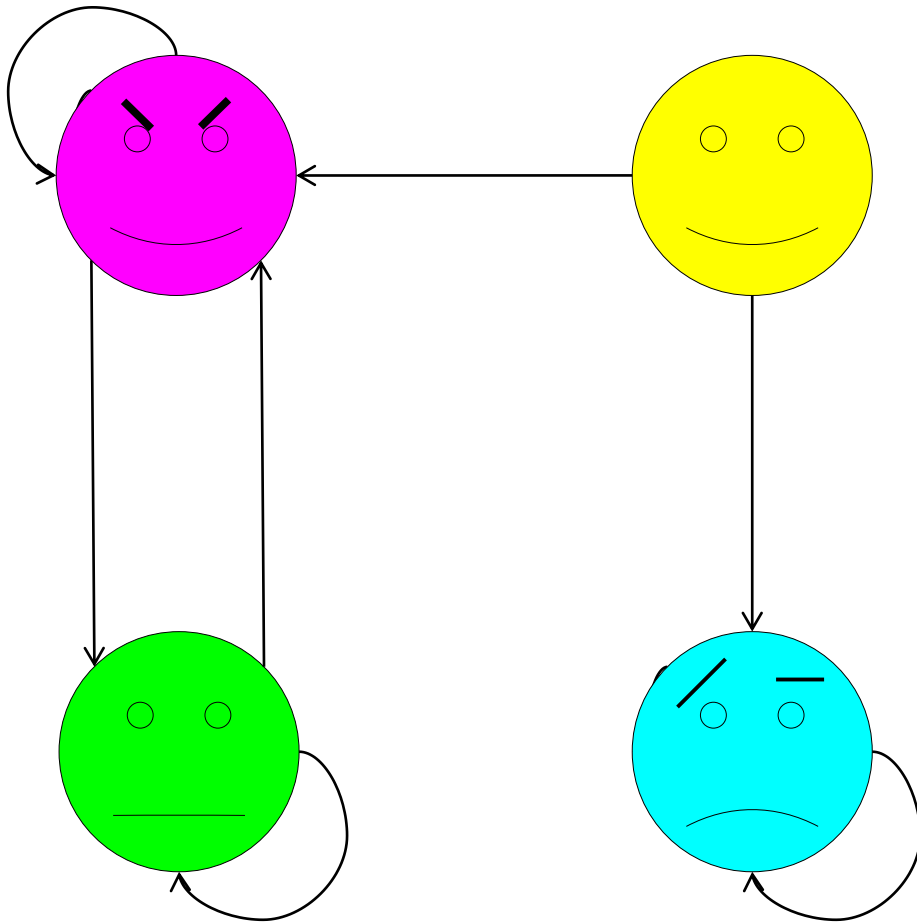
$\forall a \in A. aRa$


(“Every element is related to itself.”)



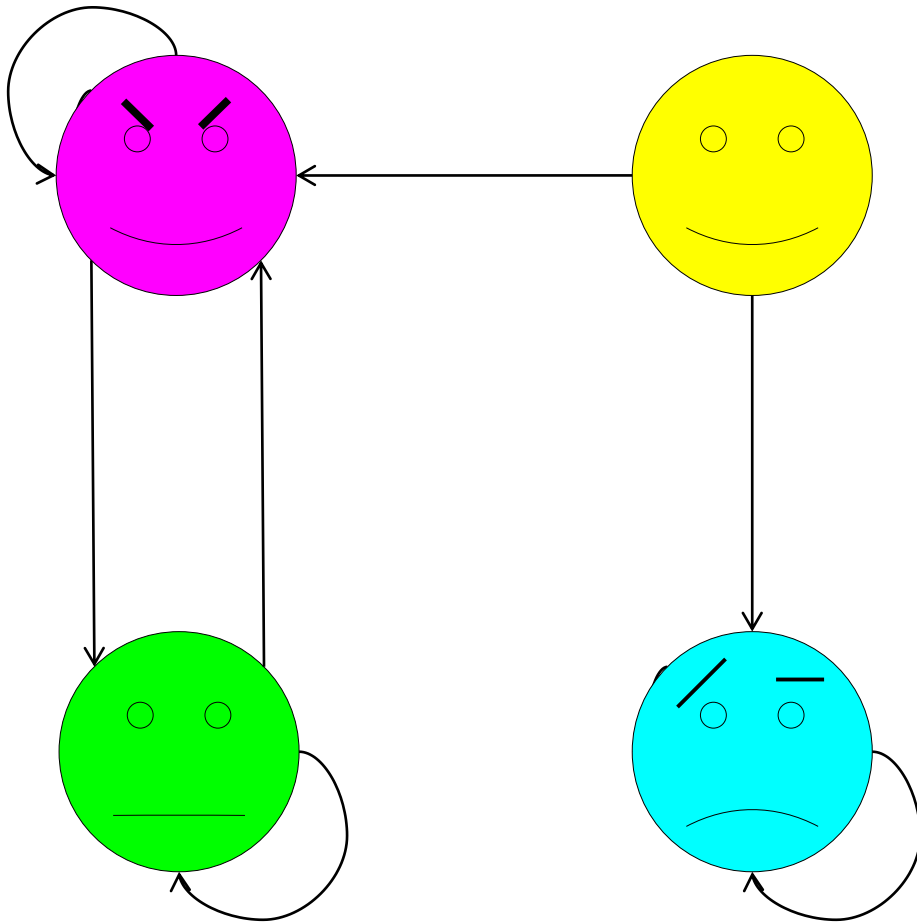
This means that R is not reflexive, since the first-order logic statement given below is not true.


$\forall a \in A. aRa$
(“Every element is related to itself.”)



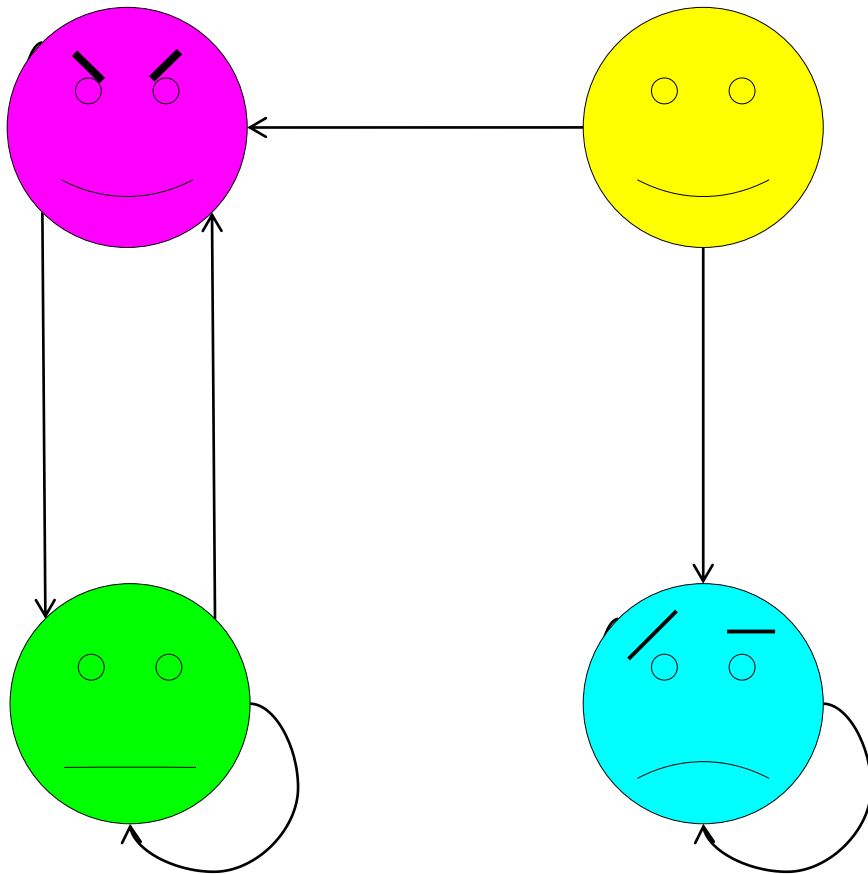
Is  reflexive?

$\forall a \in A. aRa$
(“Every element is related to itself.”)



Is  reflexive?

$\forall a \in ?? . a$  a



Is  reflexive?

Reflexivity is a property of *relations*, not *individual objects*.

$\forall a \in ?? . a \text{  } a$

$$\forall a \in A. aRa$$

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

$$\forall a \in A. aRa$$

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

Symmetry

In some relations, the relative order of the objects doesn't matter.

Examples:

- If $x = y$, then $y = x$.
- If $x \equiv_k y$, then $y \equiv_k x$.

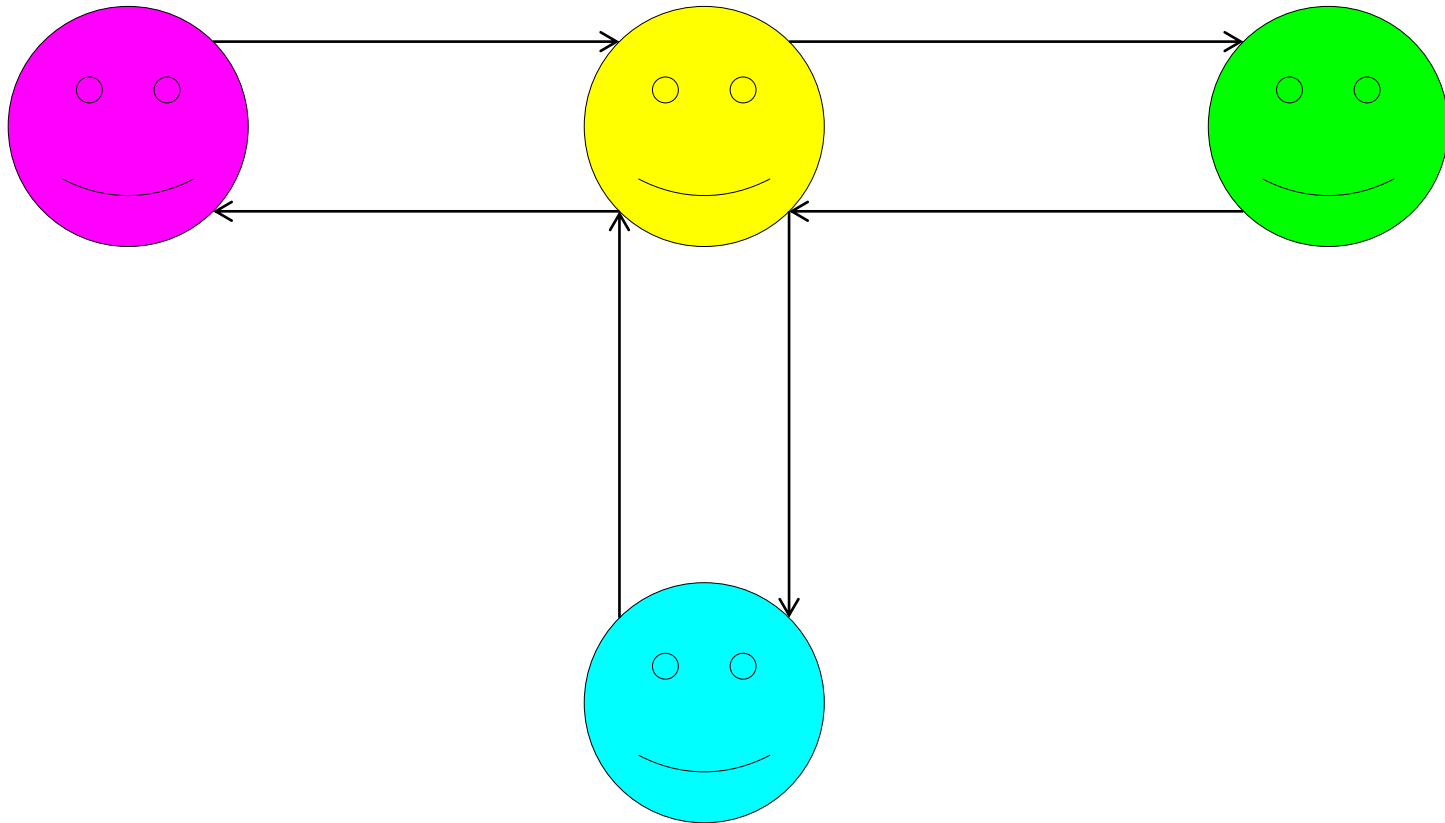
These relations are called ***symmetric***.

Formally: a binary relation R over a set A is called *symmetric* if the following first-order statement is true about R :

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

(“If a is related to b , then b is related to a .”)

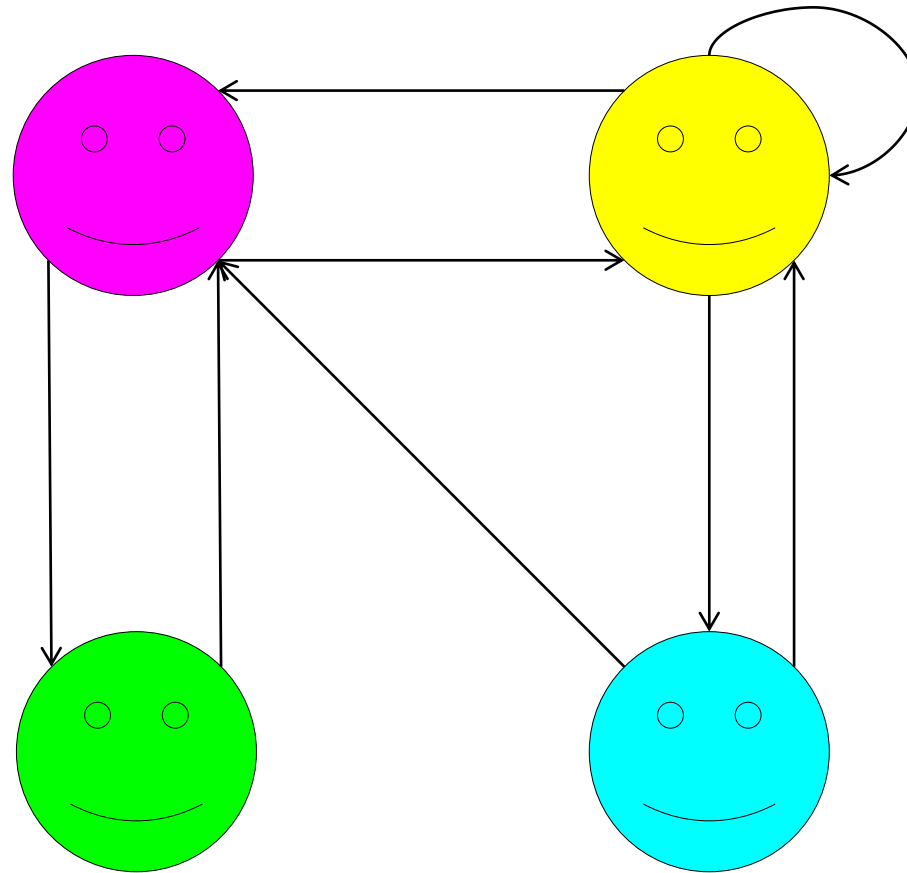
Symmetry Visualized



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

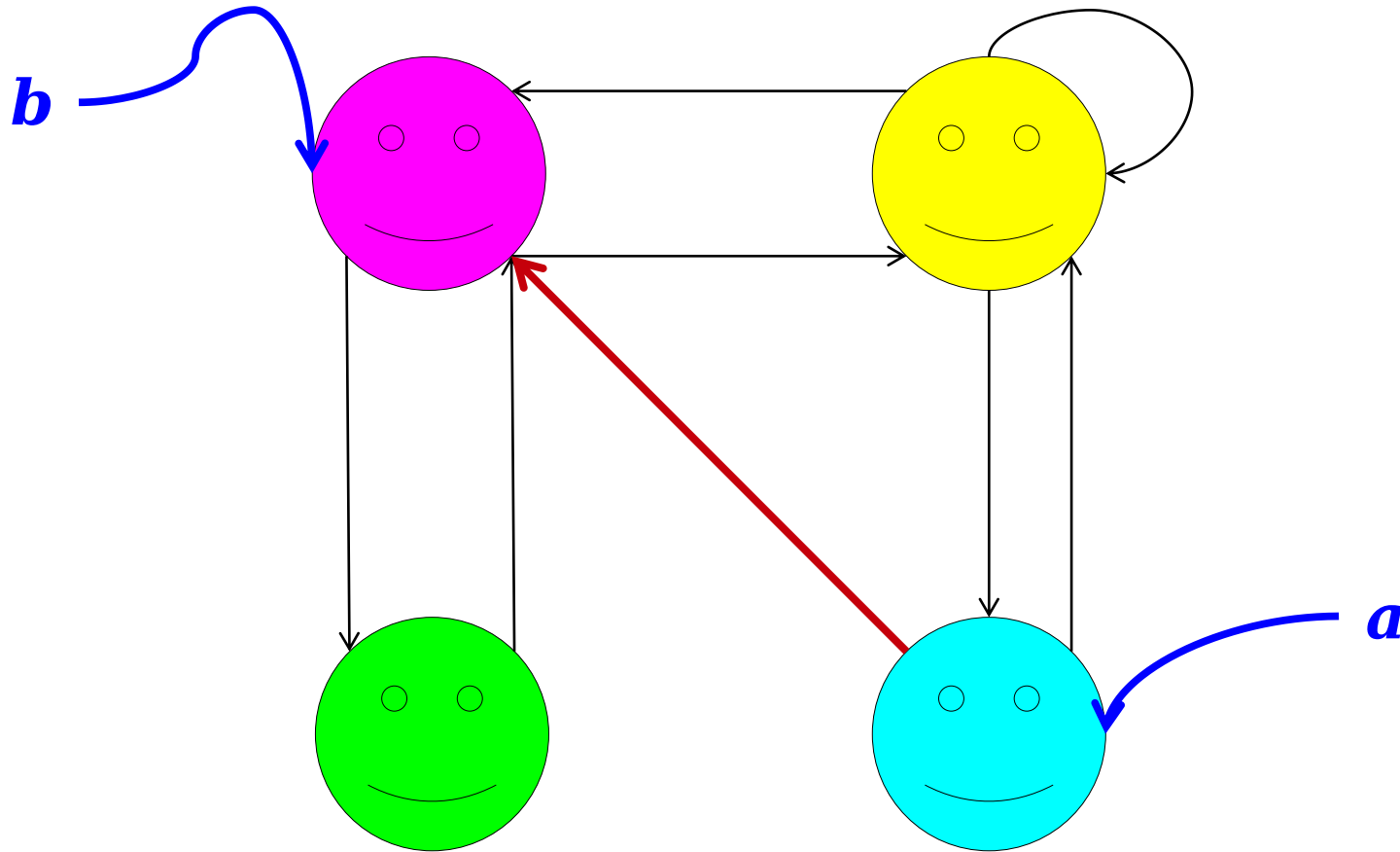
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

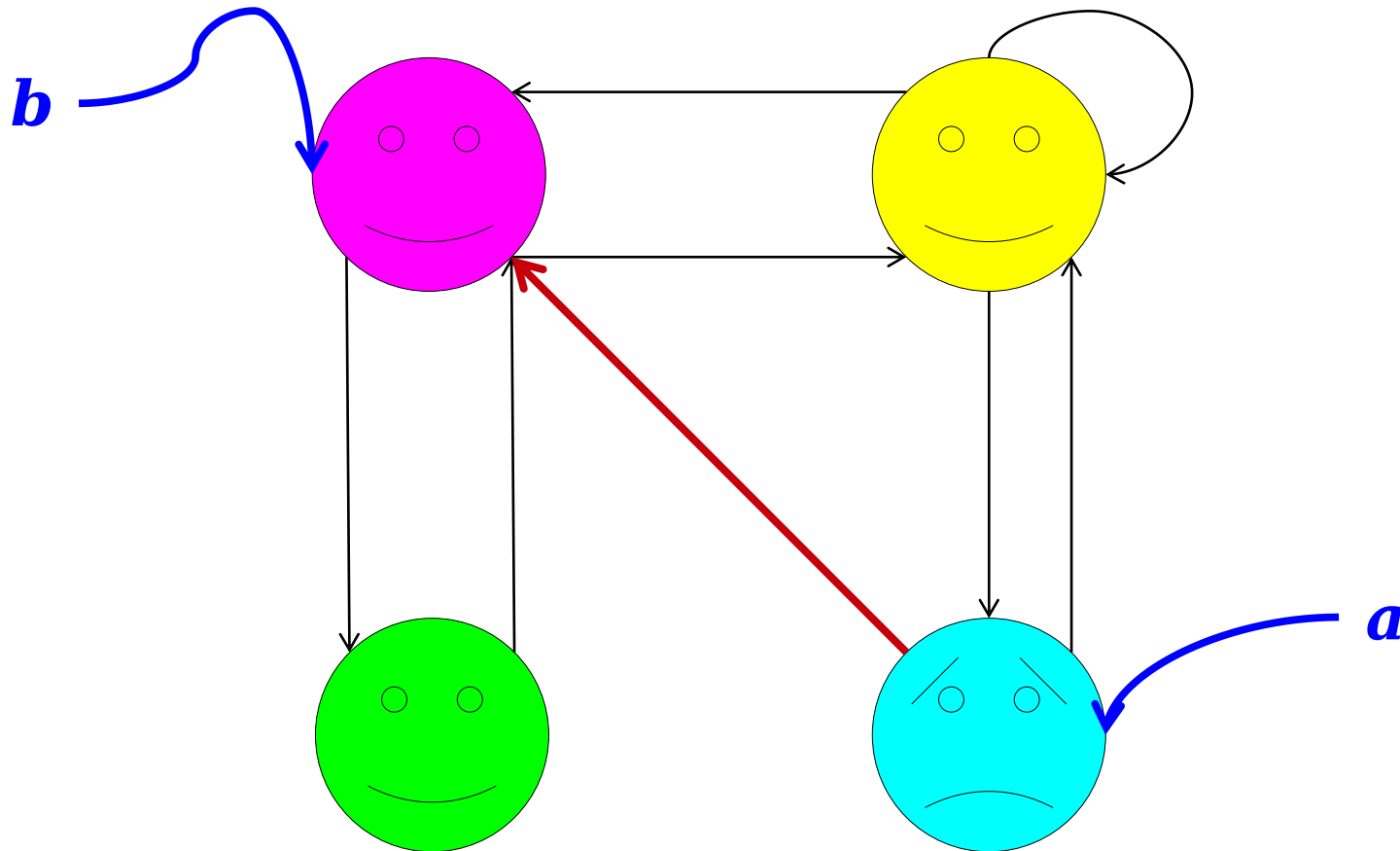
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

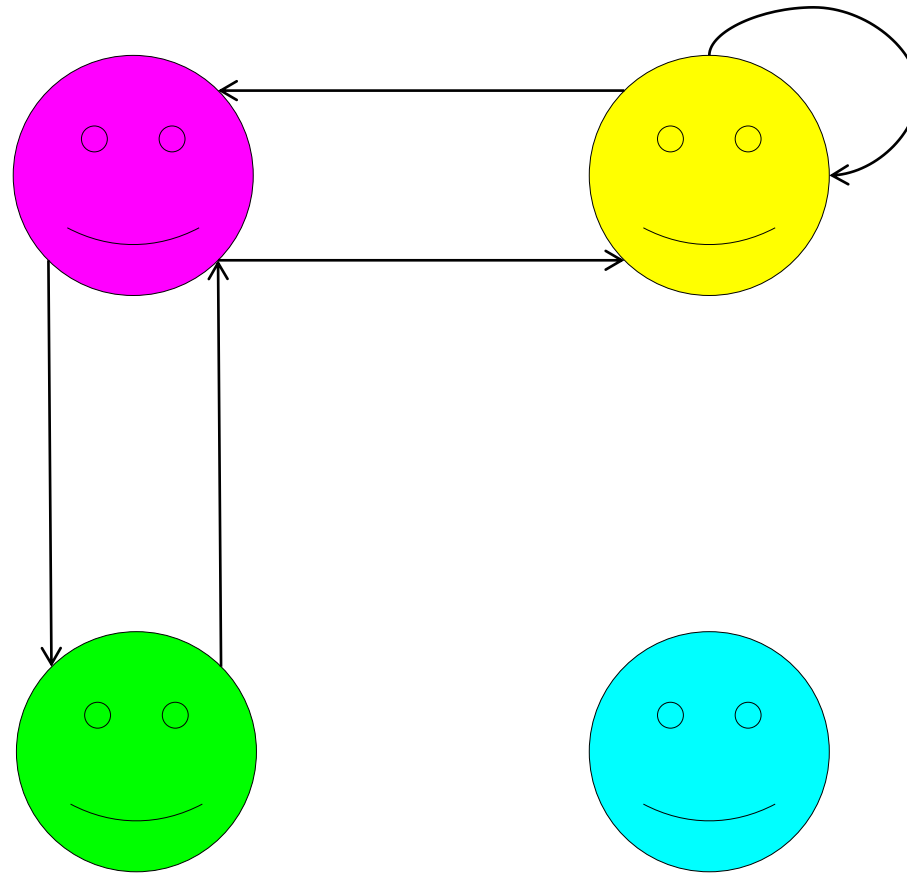
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

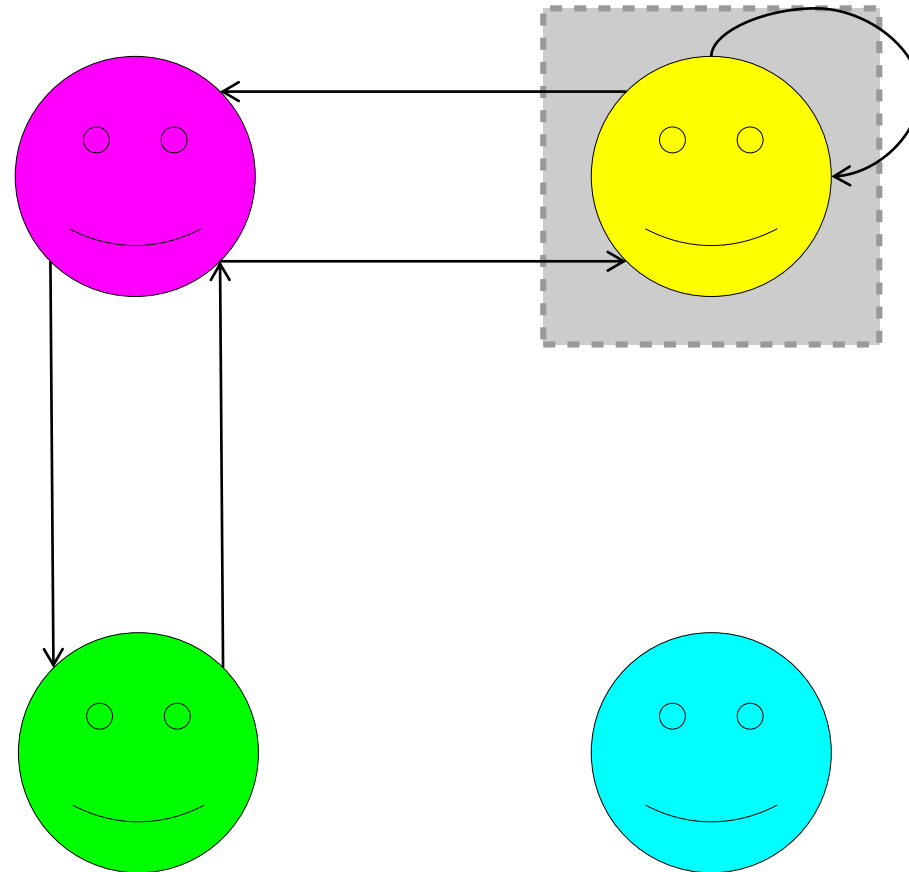
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

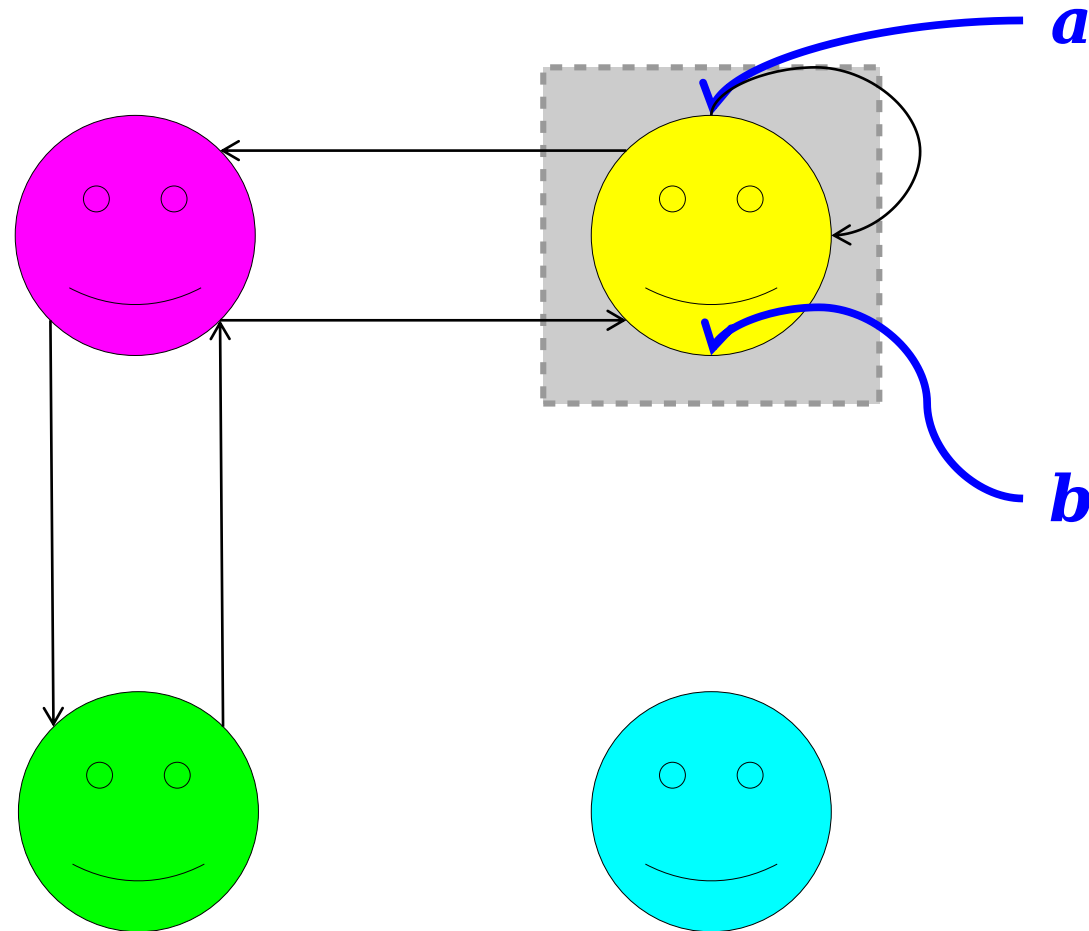
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

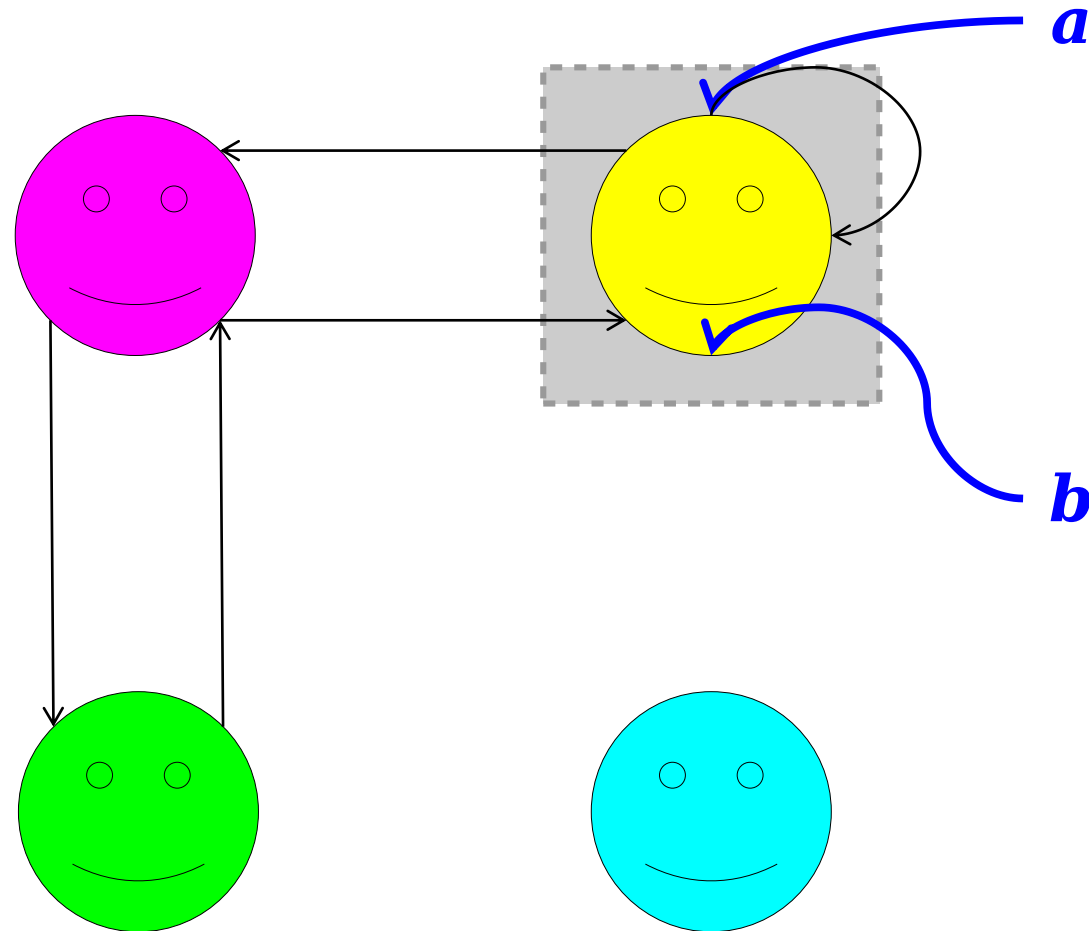
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

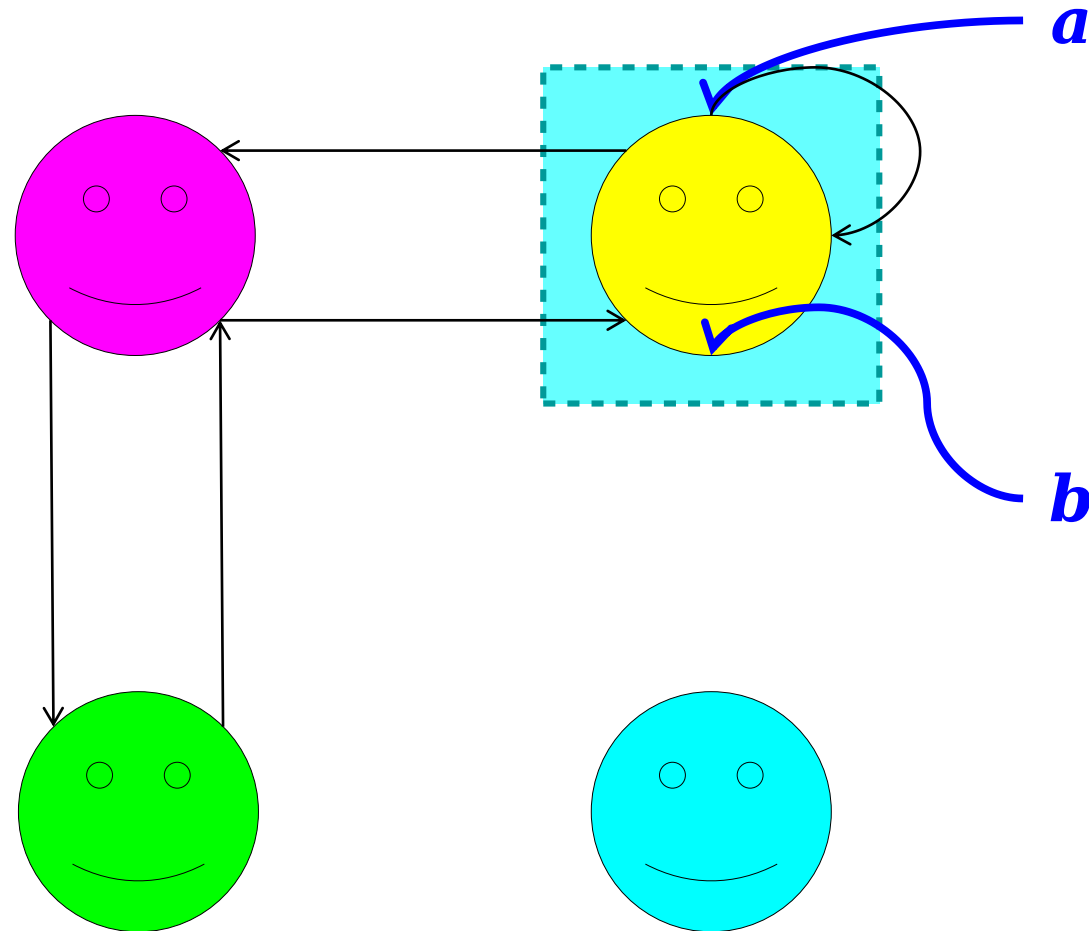
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

*(“If *a* is related to *b*, then *b* is related to *a*.”)*

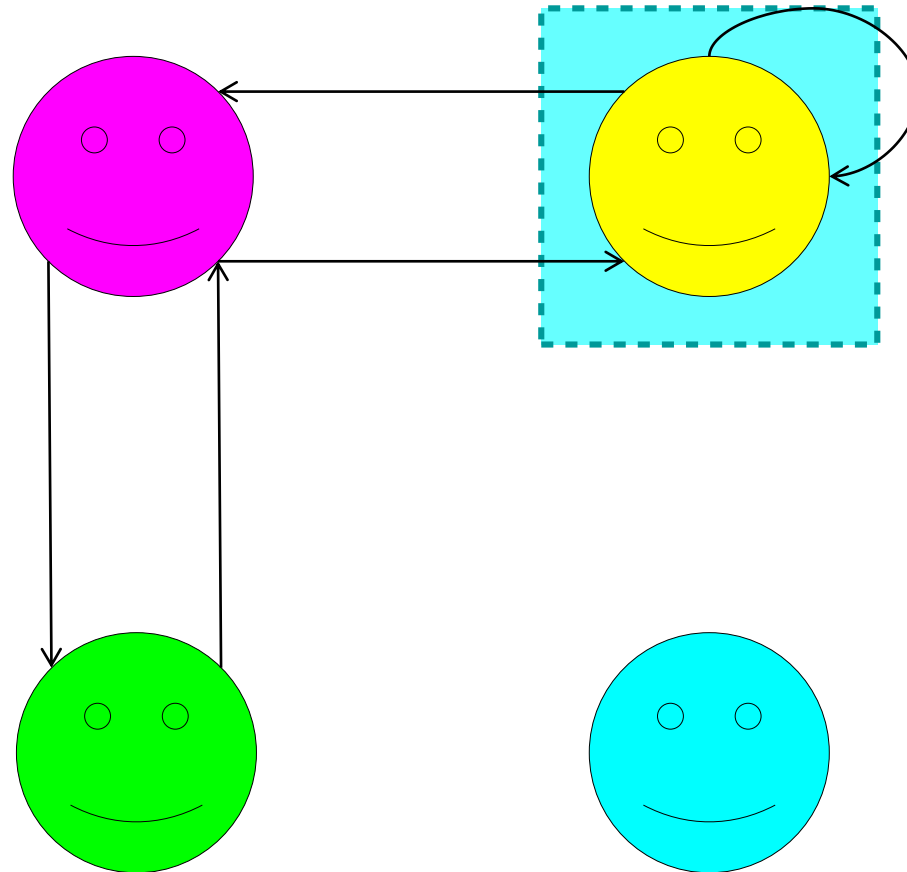
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

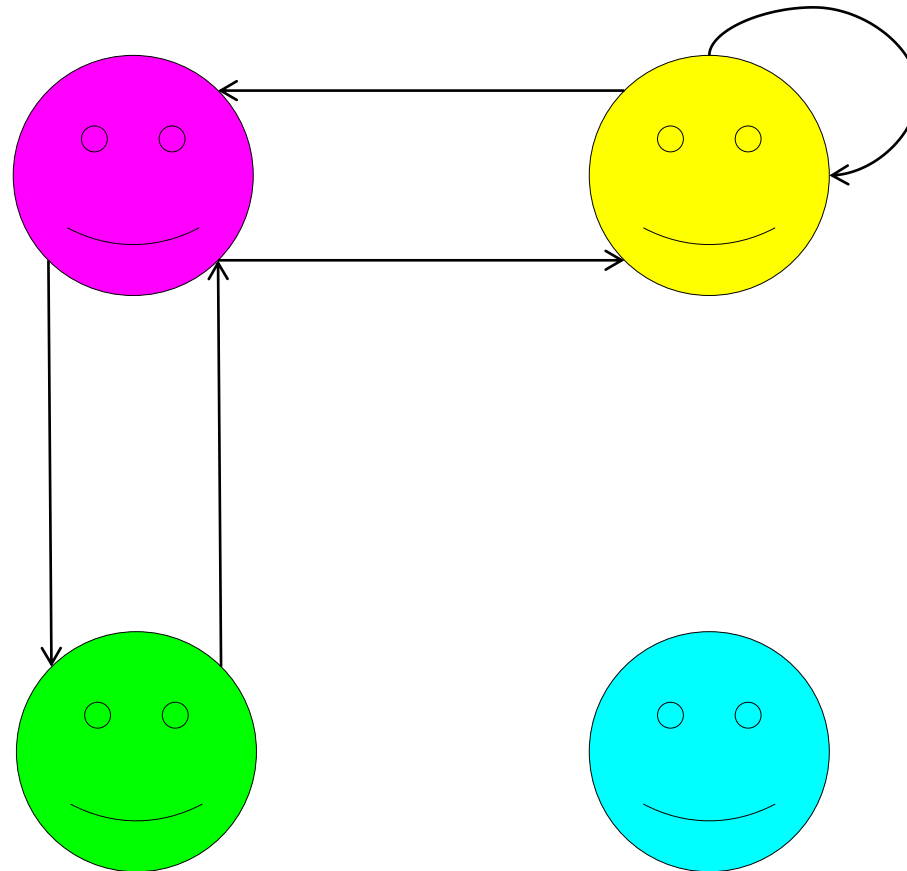
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

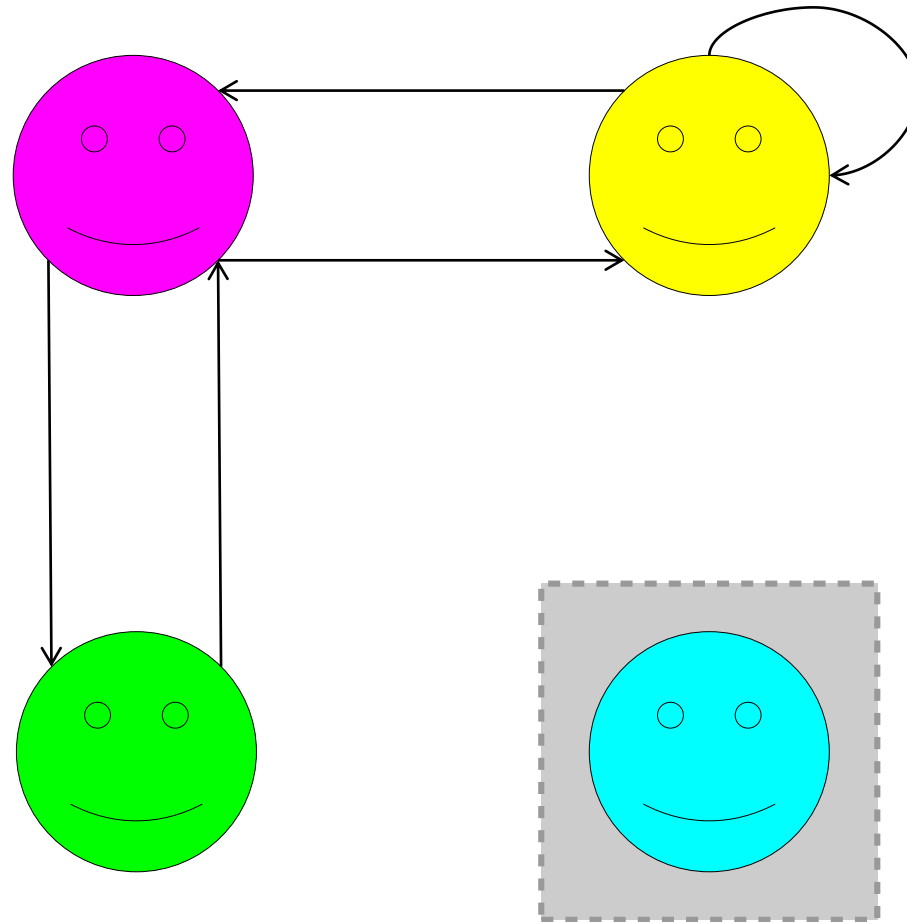
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

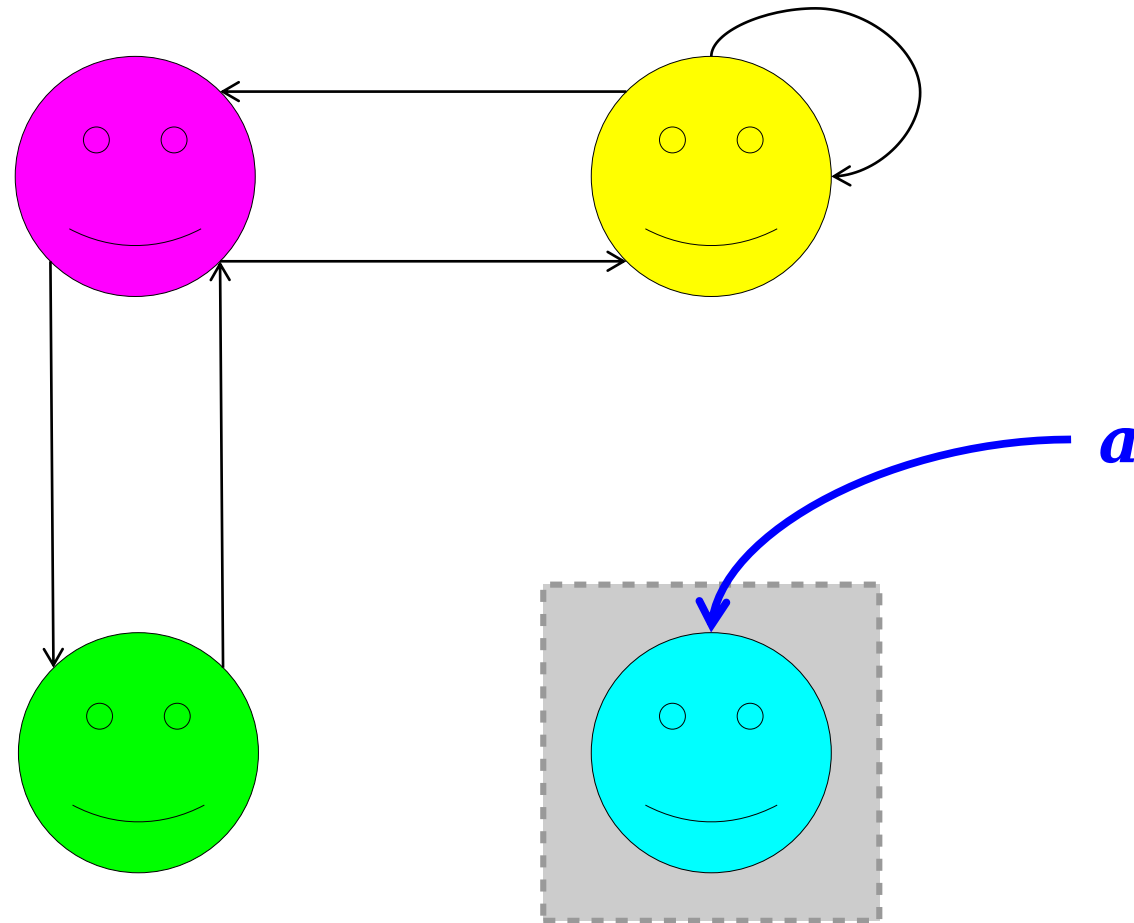
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

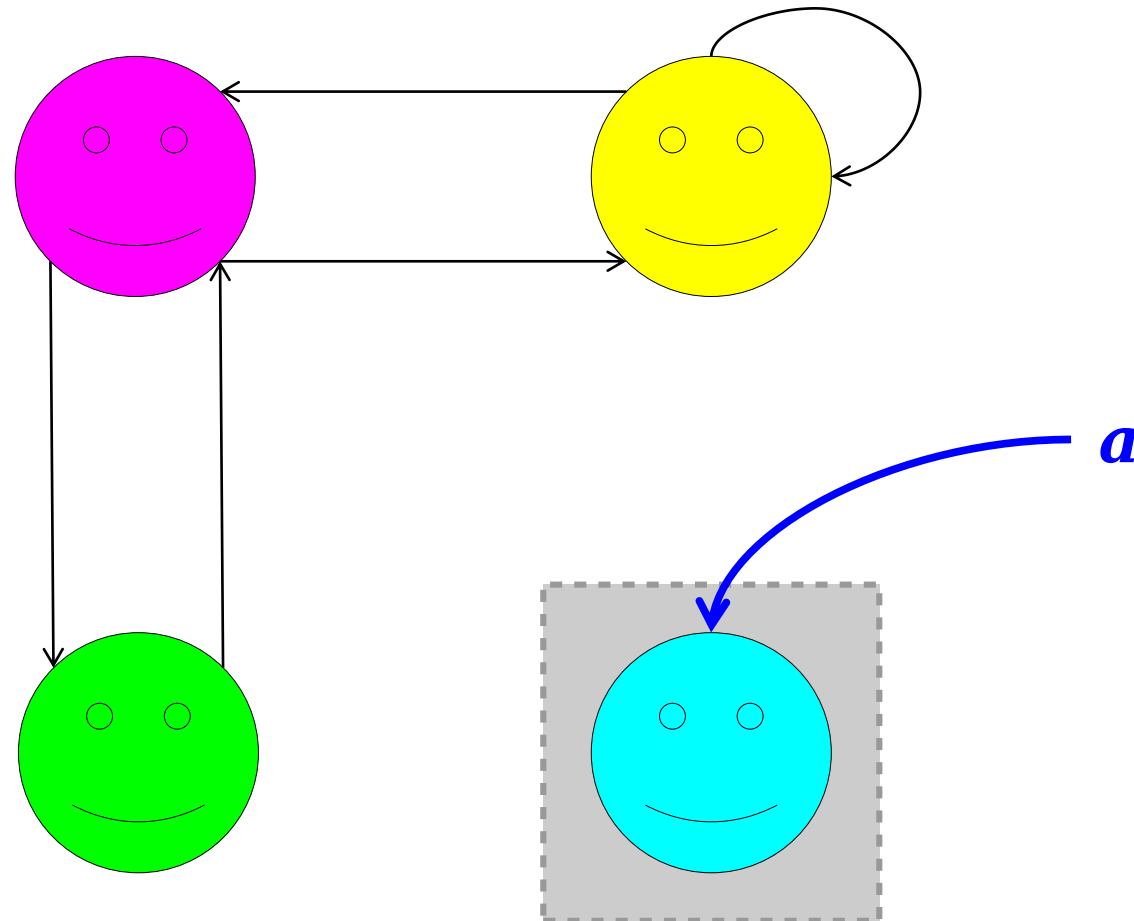
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

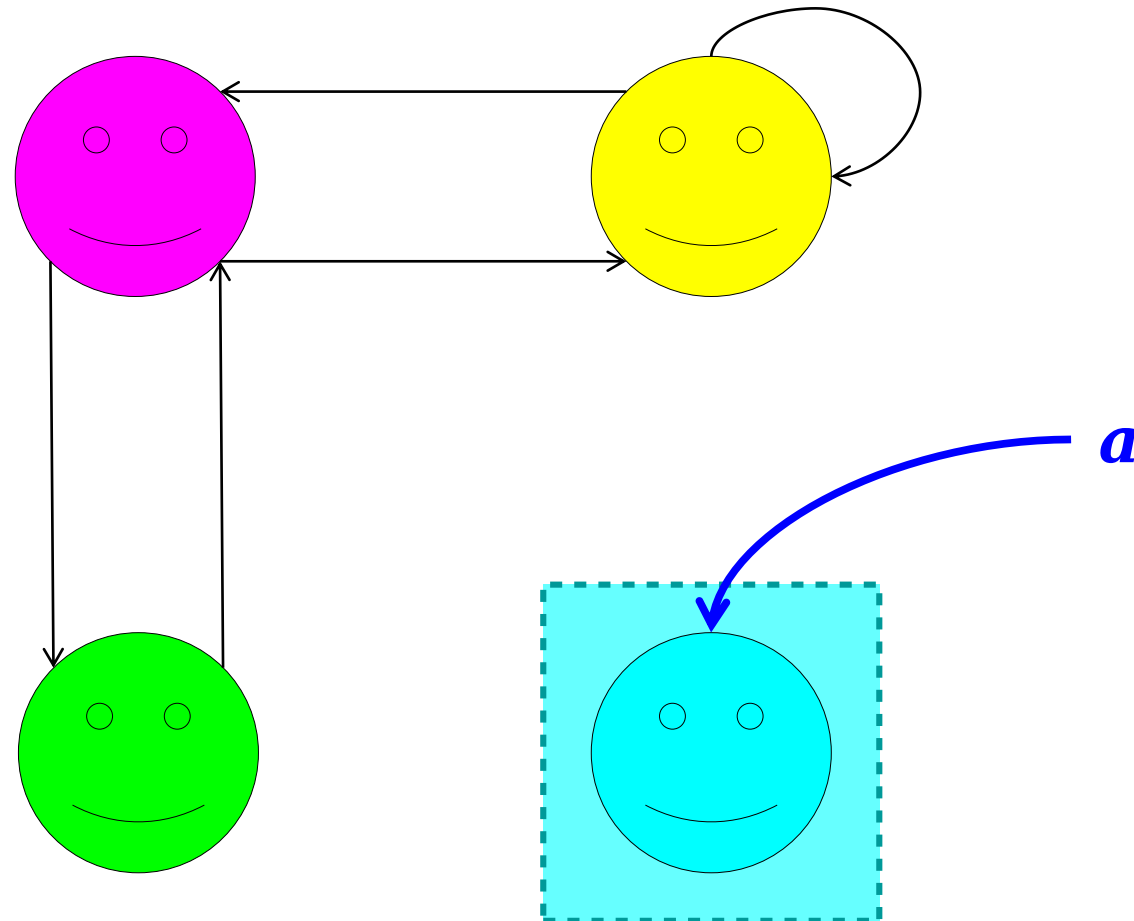
Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

Is This Relation Symmetric?



$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

(“If a is related to b , then b is related to a .”)

$$\forall a \in A. aRa$$

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

$$\forall a \in A. aRa$$

$$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

Transitivity

Many relations can be chained together.

Examples:

- If $x = y$ and $y = z$, then $x = z$.
- If $R \subseteq S$ and $S \subseteq T$, then $R \subseteq T$.
- If $x \equiv_k y$ and $y \equiv_k z$, then $x \equiv_k z$.

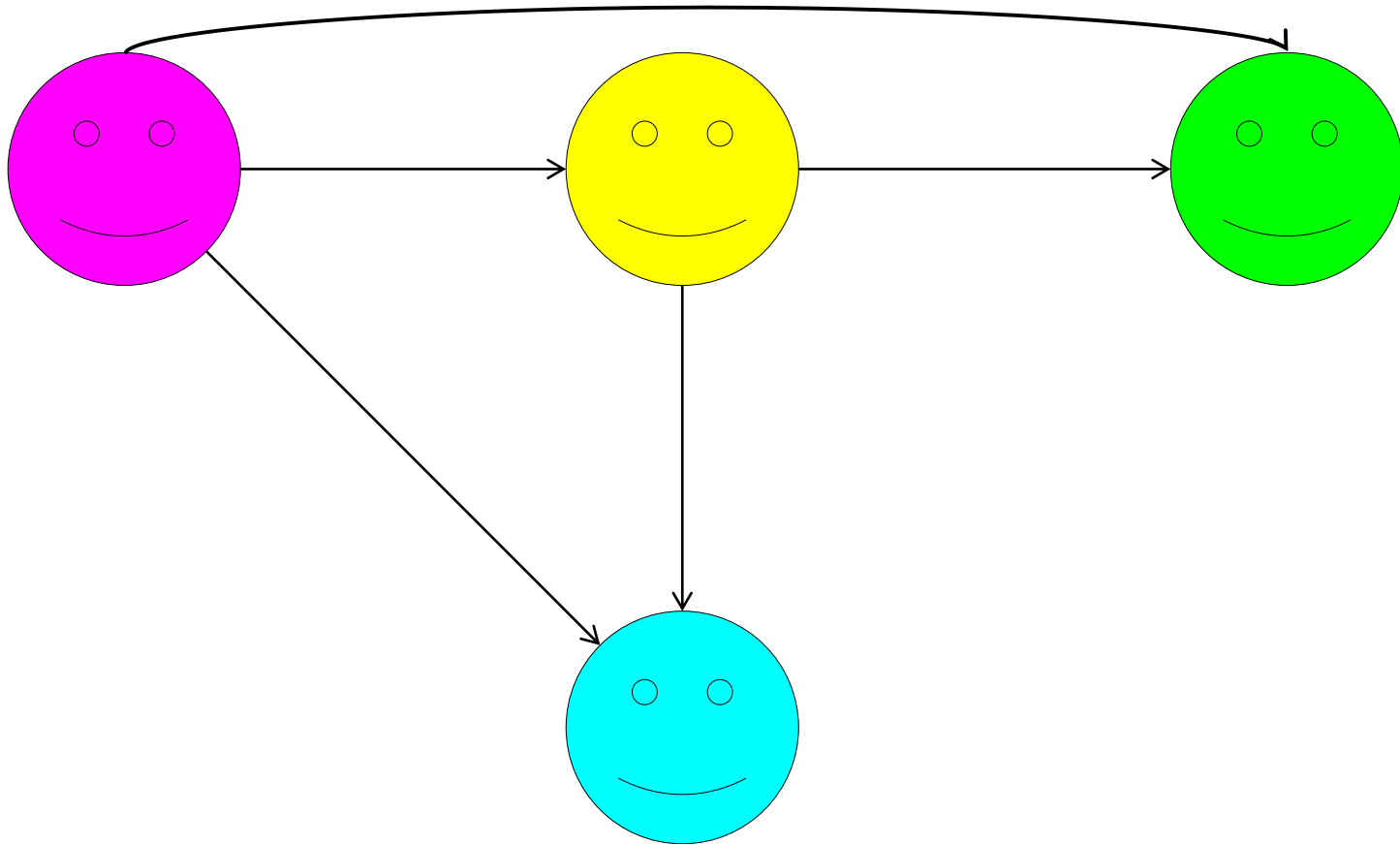
These relations are called ***transitive***.

A binary relation R over a set A is called *transitive* if the following first-order statement is true about R :

$$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$$

(“Whenever a is related to b and b is related to c , we know a is related to c .)

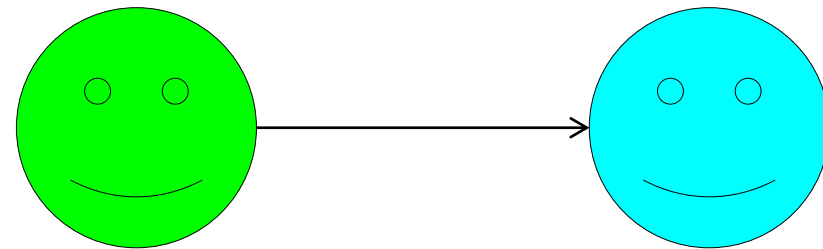
Transitivity Visualized



$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$

("Whenever a is related to b and b is related to c , we know a is related to c .)

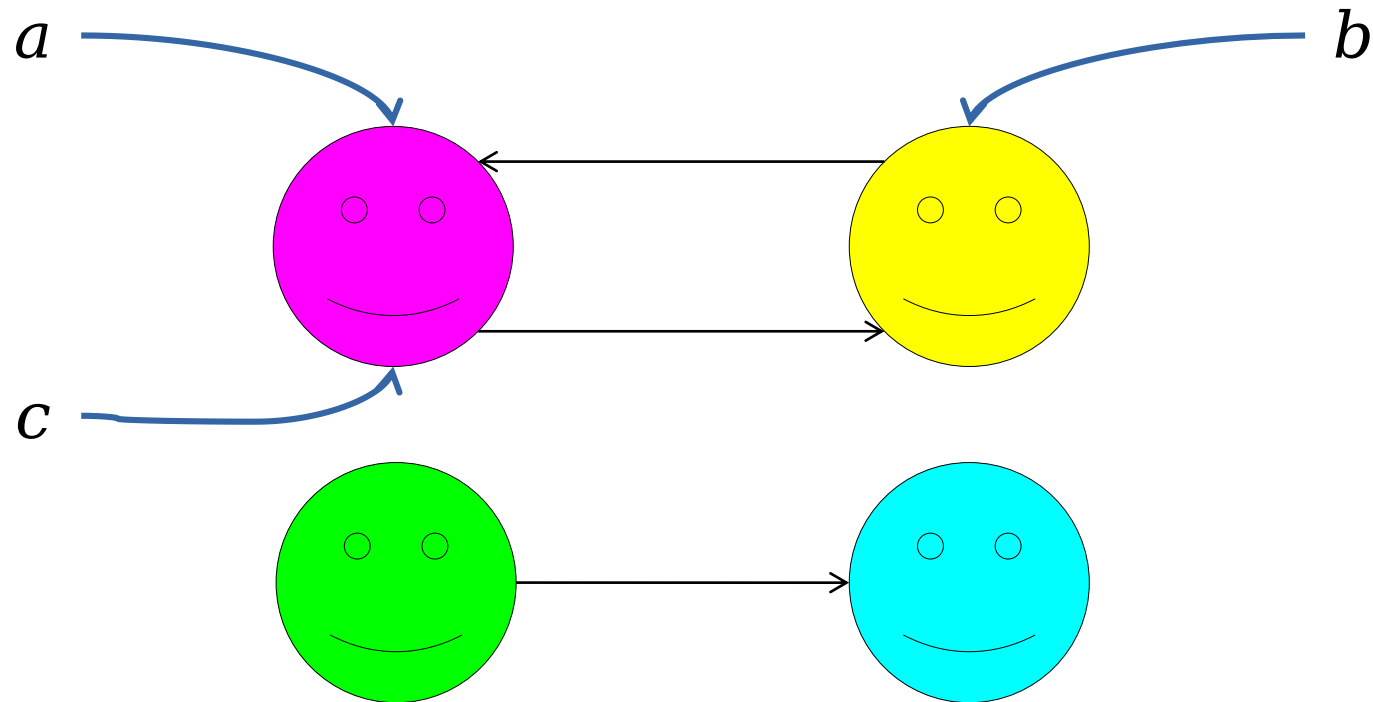
Is This Relation Transitive?



$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$

("Whenever a is related to b and b is related to c, we know a is related to c.")

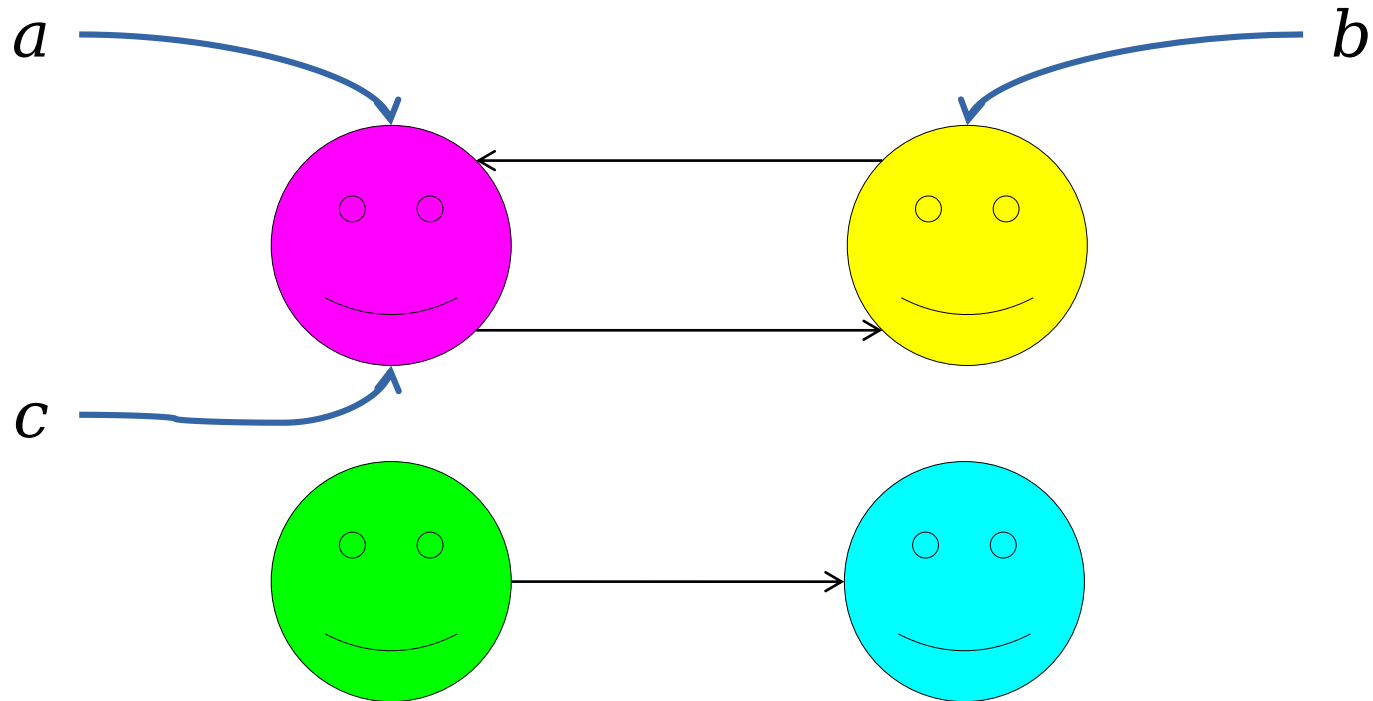
Is This Relation Transitive?



$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$

("Whenever a is related to b and b is related to c, we know a is related to c.")

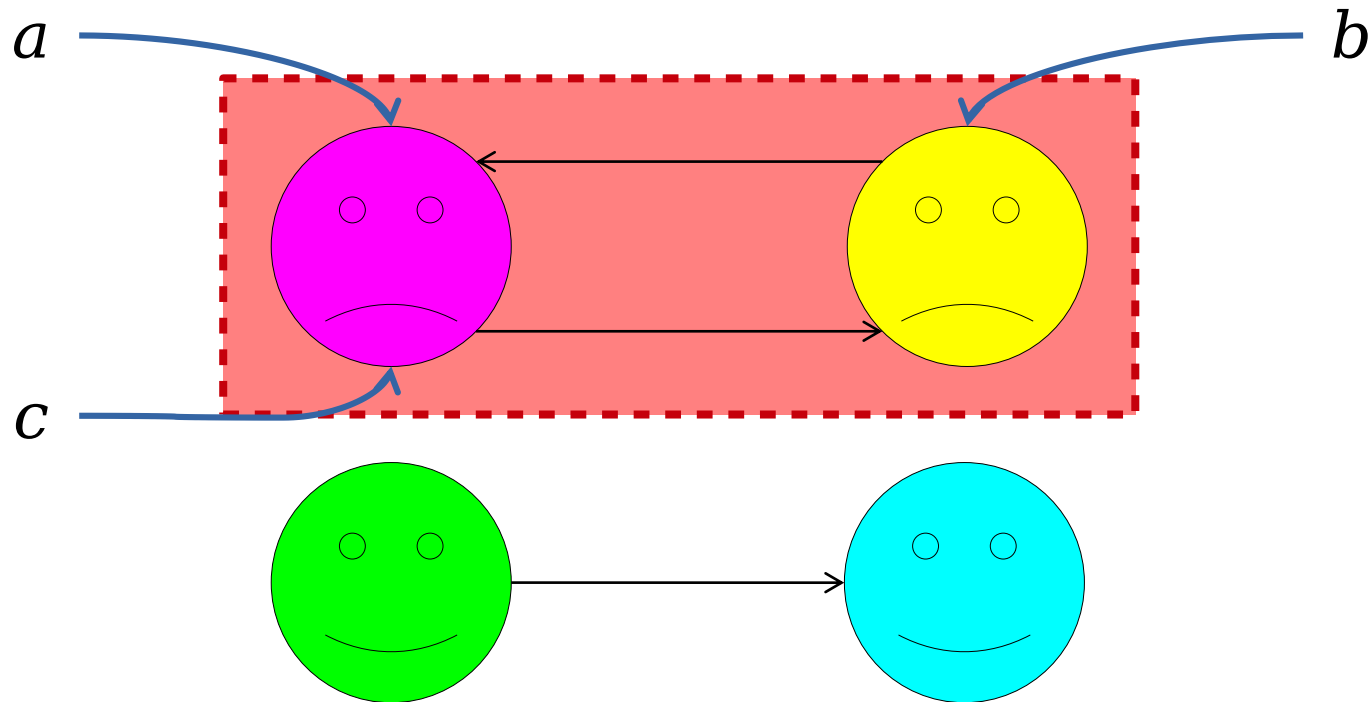
Is This Relation Transitive?



$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$

("Whenever a is related to b and b is related to c, we know a is related to c.")

Is This Relation Transitive?



$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$

("Whenever a is related to b and b is related to c , we know a is related to c .)

Equivalence Relations

An ***equivalence relation*** is a relation that is reflexive, symmetric and transitive.

Some examples:

- $x = y$
- $x \equiv_k y$
- x has the same color as y
- x has the same shape as y .

Time-Out for Announcements!

Problem Set One Grades

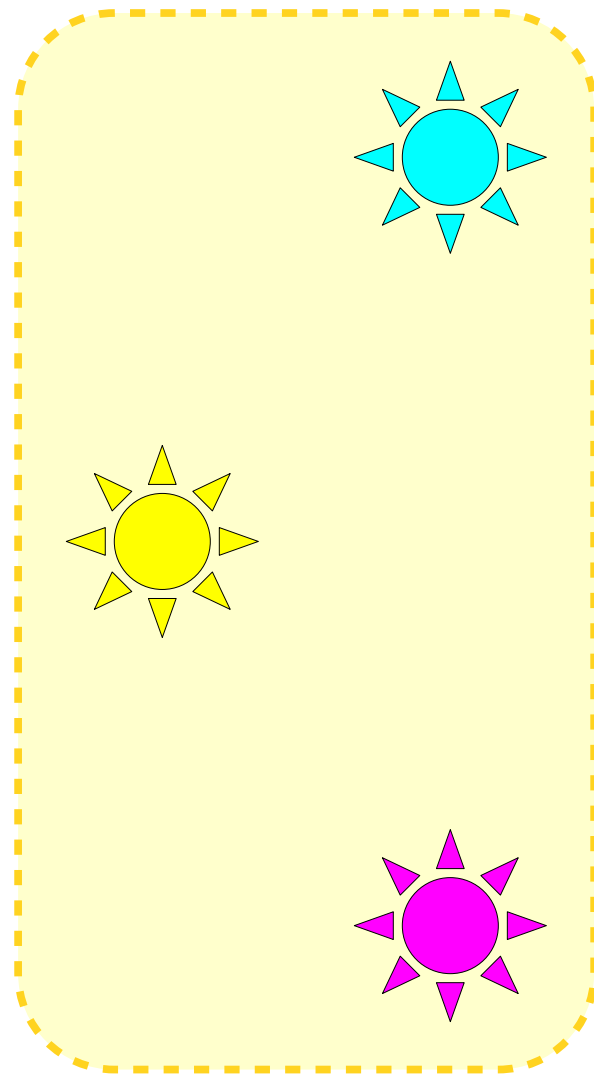
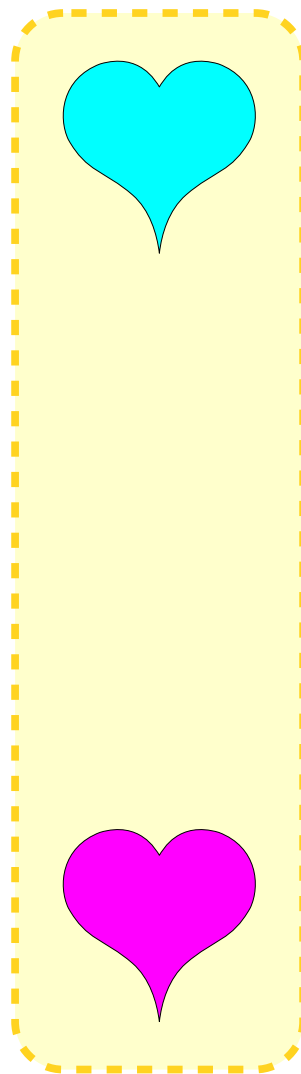
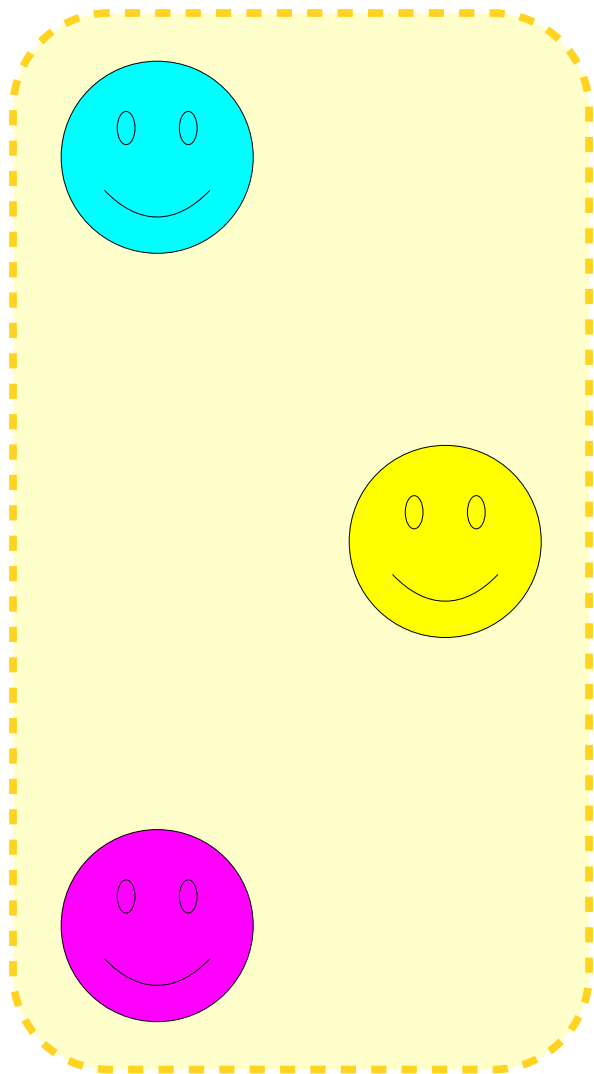
- ***You need to read over the feedback on your pset 1 as soon as possible.***
- Easiest way to improve in this course is attempting problems and incorporating feedback.
- Almost the entire point of psets is to give practice.

Problem Set Two

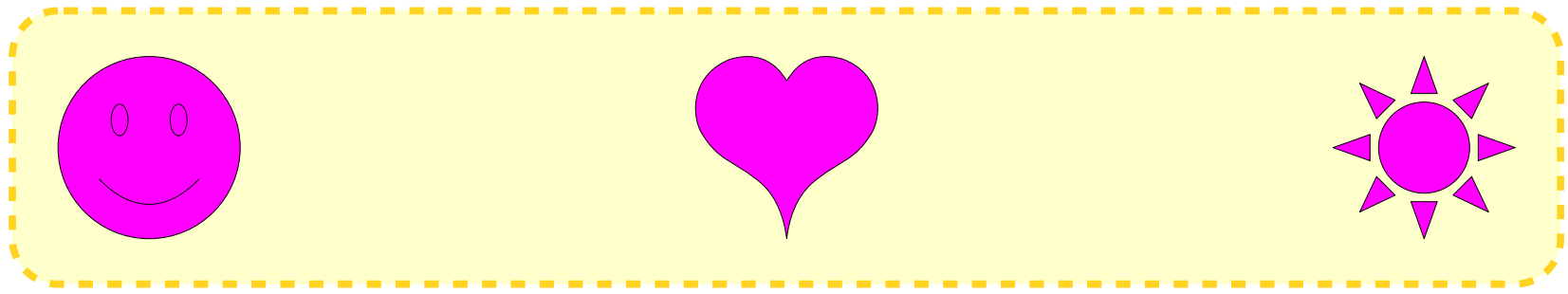
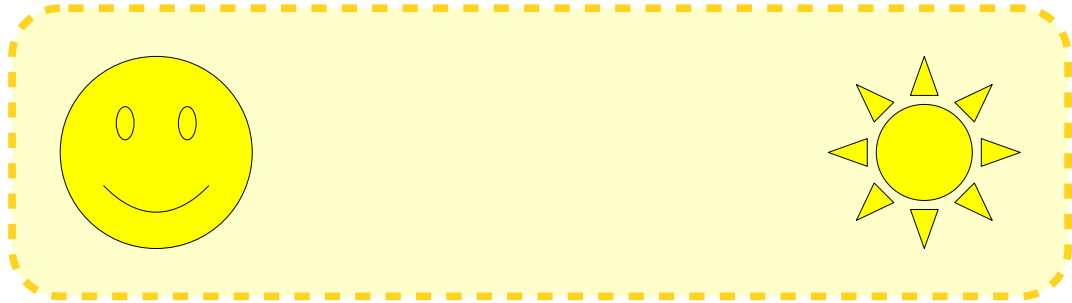
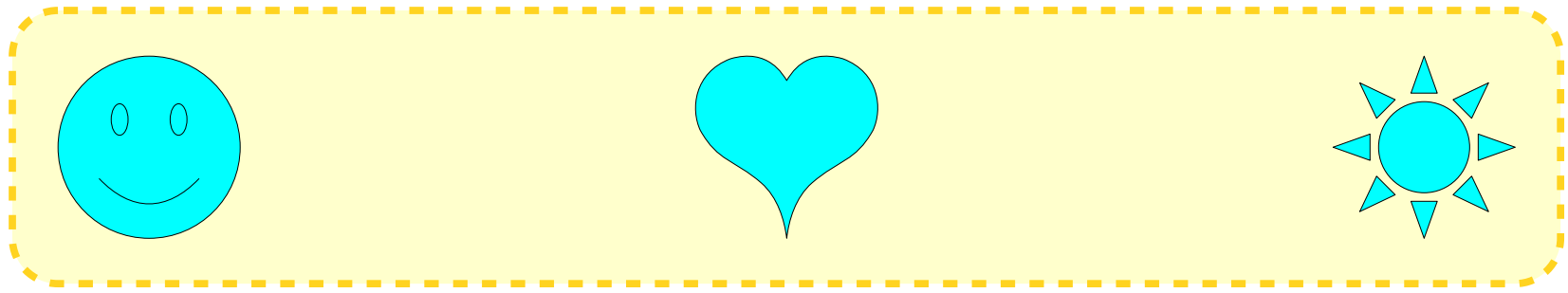
- The problem set is due Thursday at 11:59PM.
- Have questions?
- Stop by office hours!
- Ask on Campuswire!
- General problem set policy reminders:
- Please tag your pages on Gradescope
- All solutions must be typed in LaTeX.
- Partners should only make one submission - put both partners names on the PDF and tag both partners on Gradescope
- Working in partners is encouraged!

Back to CS103!

What's the connection between partitions
and binary relations?



xRy if x and y have the same shape



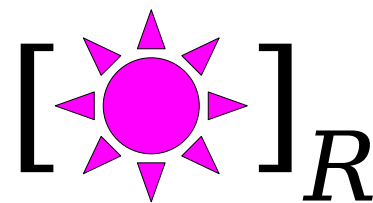
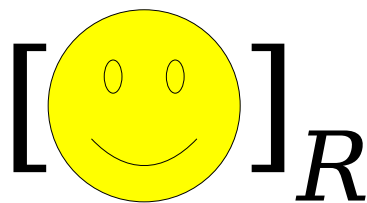
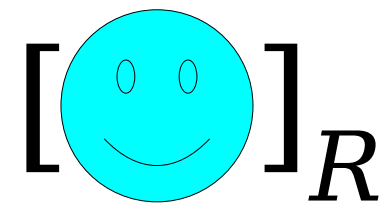
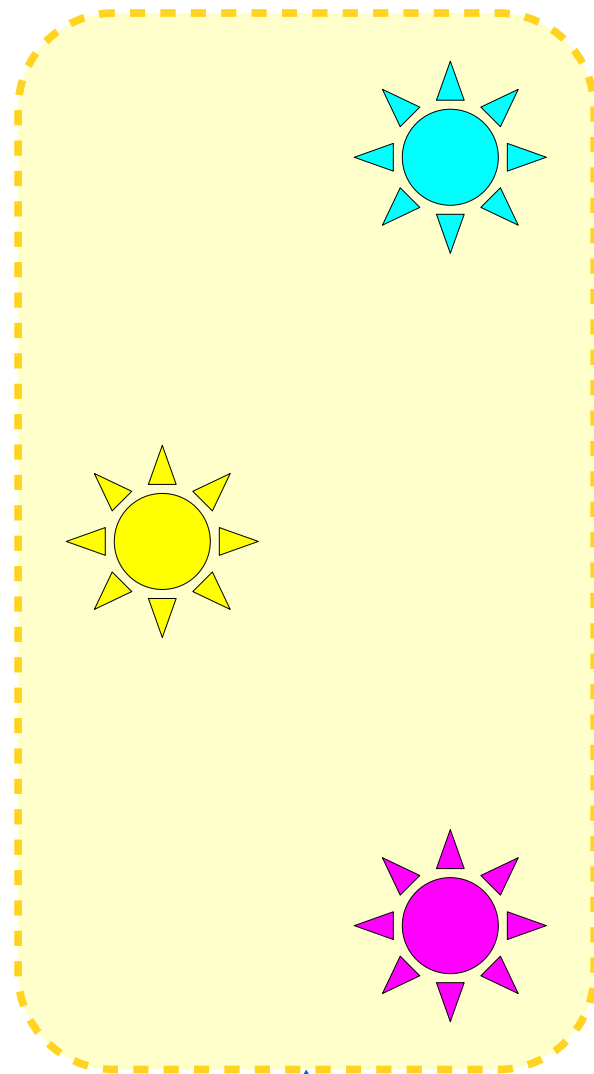
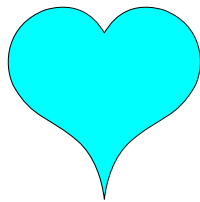
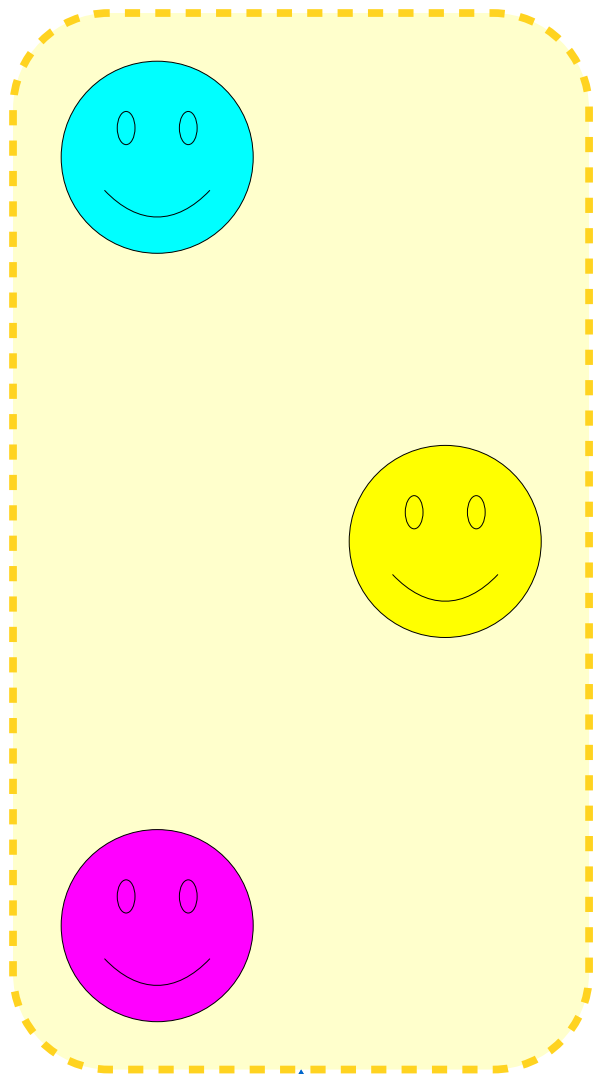
xTy if x and y have the same color

Equivalence Classes

Given an equivalence relation R over a set A , for any $x \in A$, the **equivalence class of x** is the set

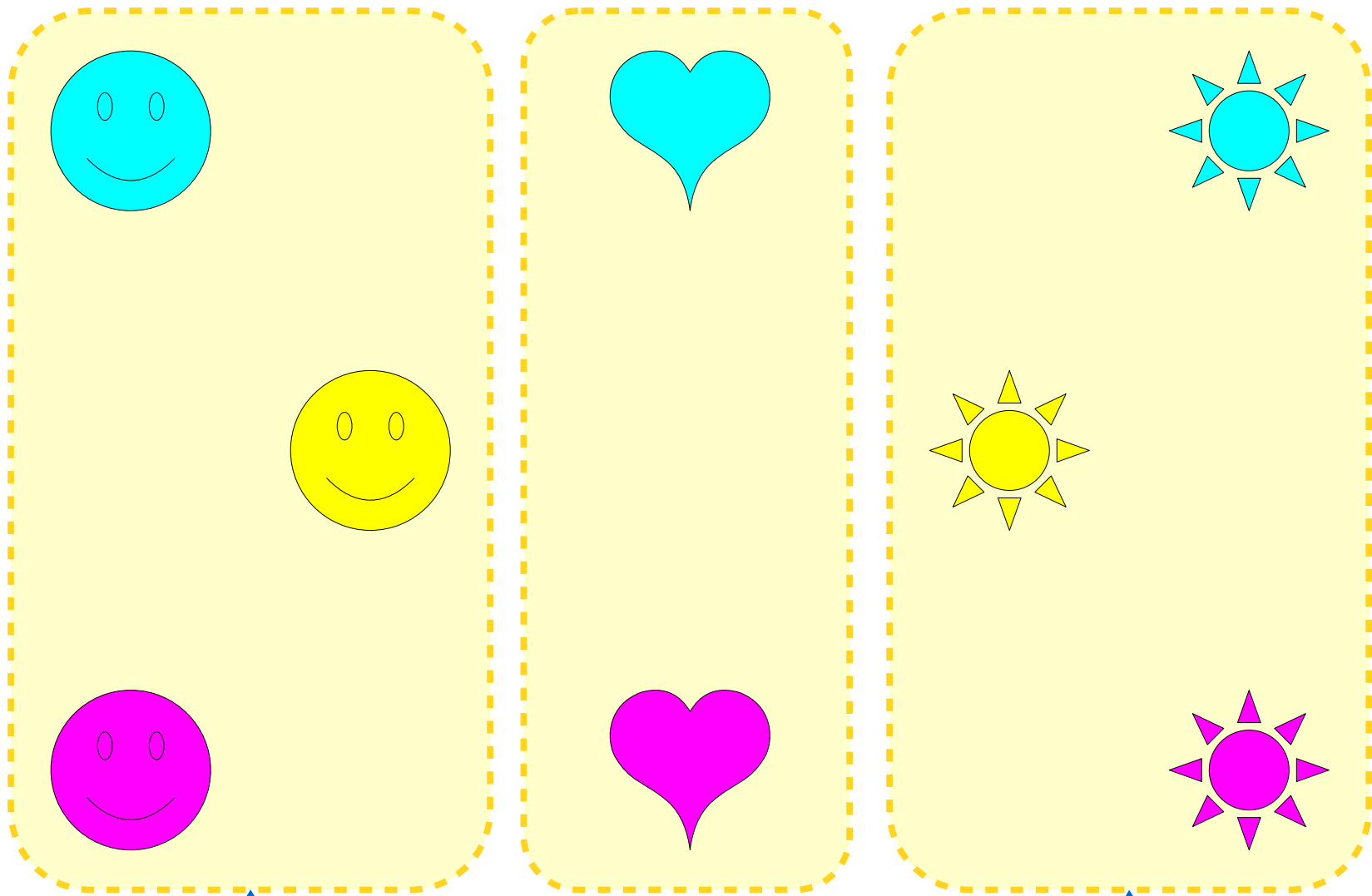
$$[x]_R = \{ y \in A \mid xRy \}$$

Intuitively, the set $[x]_R$ contains all elements of A that are related to x by relation R .



xRy if x and y have the same shape

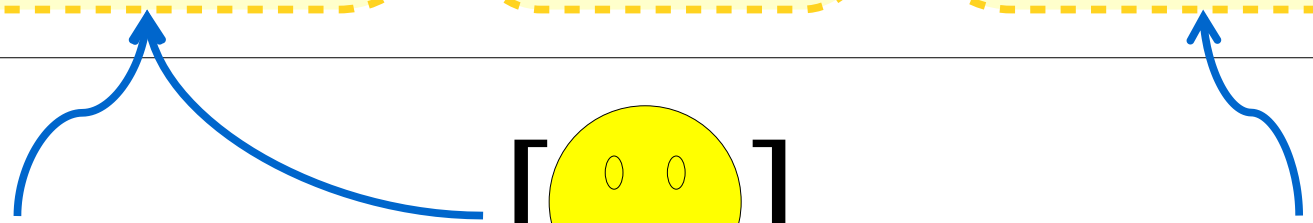
***The Fundamental Theorem of
Equivalence Relations:*** Let R be an
equivalence relation over a set A . Then
every element $a \in A$ belongs to exactly one
equivalence class of R .



$$[\text{smiley face}]_R$$

$$[\text{smiley face}]_R$$

$$[\text{sun}]_R$$



xRy if x and y have the same shape

How'd We Get Here?

We discovered equivalence relations by thinking about *partitions* of a set of elements.

We saw that if we had a binary relation that tells us whether two elements are in the same group, it had to be reflexive, symmetric, and transitive.

The FToER says that, in some sense, these rules precisely capture what it means to be a partition.

Binary relations give us a ***common language*** to describe ***common structures***.

Equivalence Relations IRL

Most modern programming languages include some sort of hash table data structure.

- Java: `HashMap`
- C++: `std::unordered_map`
- Python: `dict`

If you insert a key/value pair and then try to look up a key, the implementation has to be able to tell whether two keys are equal.

Although each language has a different mechanism for specifying this, many languages describe them in similar ways...

Equivalence Relations IRL

“The equals method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value x , $x.equals(x)$ should return true.
- It is *symmetric*: for any non-null reference values x and y , $x.equals(y)$ should return true if and only if $y.equals(x)$ returns true.
- It is *transitive*: for any non-null reference values x , y , and z , if $x.equals(y)$ returns true and $y.equals(z)$ returns true, then $x.equals(z)$ should return true.”

Java 8 Documentation

Equivalence Relations IRL

“The equals method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value x , $x.equals(x)$ should return true.
- It is *symmetric*: for any non-null reference values x and y , $x.equals(y)$ should return true if and only if $y.equals(x)$ returns true.
- It is *transitive*: for any non-null reference values x , y , and z , if $x.equals(y)$ returns true and $y.equals(z)$ returns true, then $x.equals(z)$ should return true.”

Java 8 Documentation

Equivalence Relations IRL

“Each unordered associative container is parameterized by `Key`, by a function object type `Hash` that meets the Hash requirements (17.6.3.4) and acts as a hash function for argument values of type `Key`, and by a binary predicate `Pred` that induces an equivalence relation on values of type `Key`. Additionally, `unordered_map` and `unordered_multimap` associate an arbitrary mapped type `T` with the `Key`.”

C++14 ISO Spec, §23.2.5/3

Equivalence Relations IRL

“Each unordered associative container is parameterized by `Key`, by a function object type `Hash` that meets the Hash requirements (17.6.3.4) and acts as a hash function for argument values of type `Key`, **and by a binary predicate `Pred` that induces an equivalence relation on values of type `Key`**. Additionally, `unordered_map` and `unordered_multimap` associate an arbitrary mapped type `T` with the `Key`.”

C++14 ISO Spec, §23.2.5/3

Prerequisite Structures

The CS Core

CS106B
**Programming
Abstractions**

CS103
**Mathematical
Foundations of
Computing**

CS107
**Computer
Organization and
Systems**

CS109
**Intro to Probability
for Computer
Scientists**

CS110
**Principles of
Computer Systems**

CS161
**Design and Analysis
of Algorithms**

Theory



Pancakes

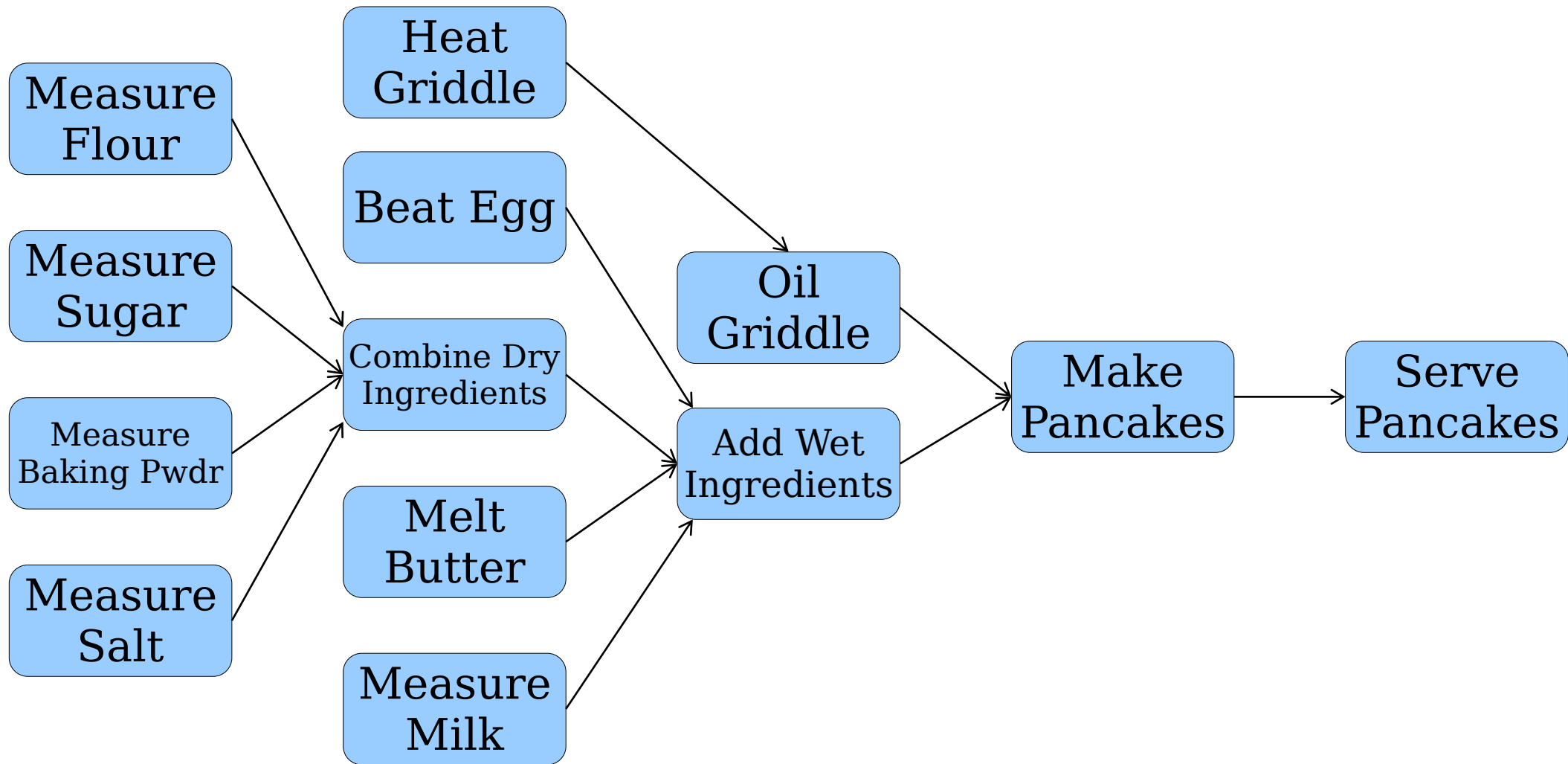
Everyone's got a pancake recipe. This one comes from Food Wishes (<http://foodwishes.blogspot.com/2011/08/grandma-kellys-good-old-fashioned.html>).

Ingredients

- 1 1/2 cups all-purpose flour
- 3 1/2 tsp baking powder
- 1 tsp salt
- 1 tbsp sugar
- 1 1/4 cup milk
- 1 egg
- 3 tbsp butter, melted

Directions

1. Sift the dry ingredients together.
2. Stir in the butter, egg, and milk. Whisk together to form the batter.
3. Heat a large pan or griddle on medium-high heat. Add some oil.
4. Make pancakes one at a time using 1/4 cup batter each. They're ready to flip when the centers of the pancakes start to bubble.



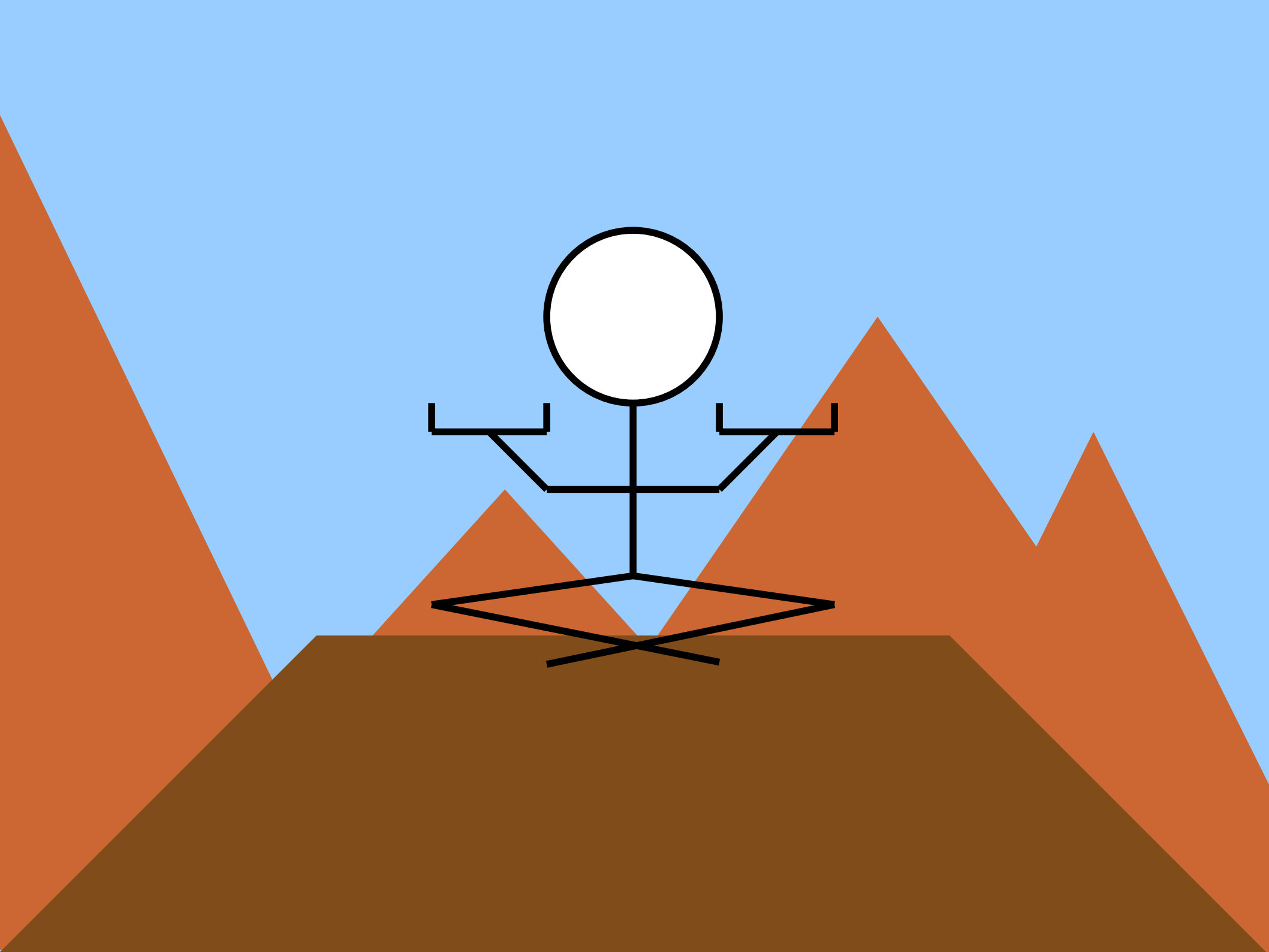
Relations and Prerequisites

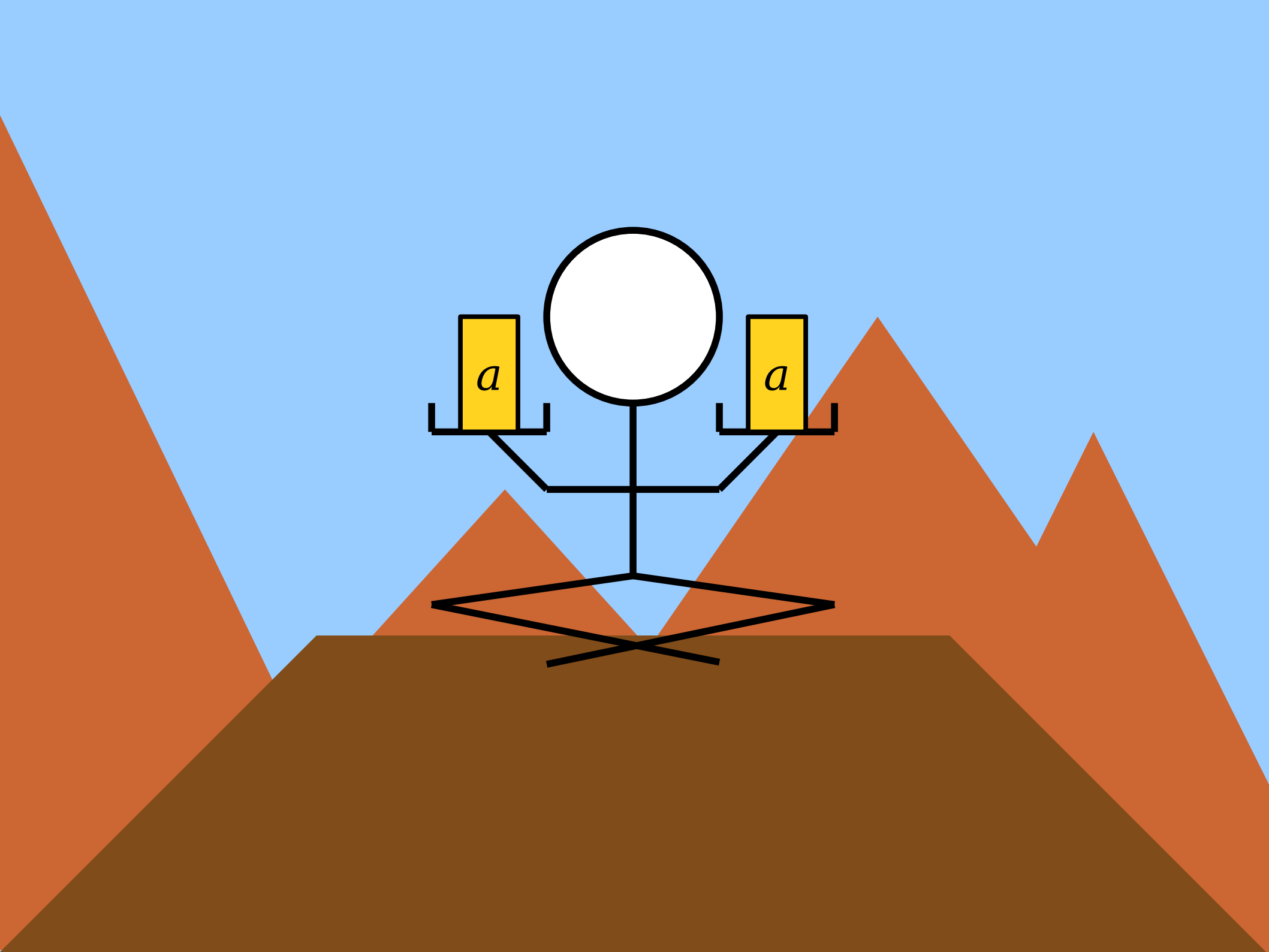
Let's imagine that we have a prerequisite structure with no circular dependencies.

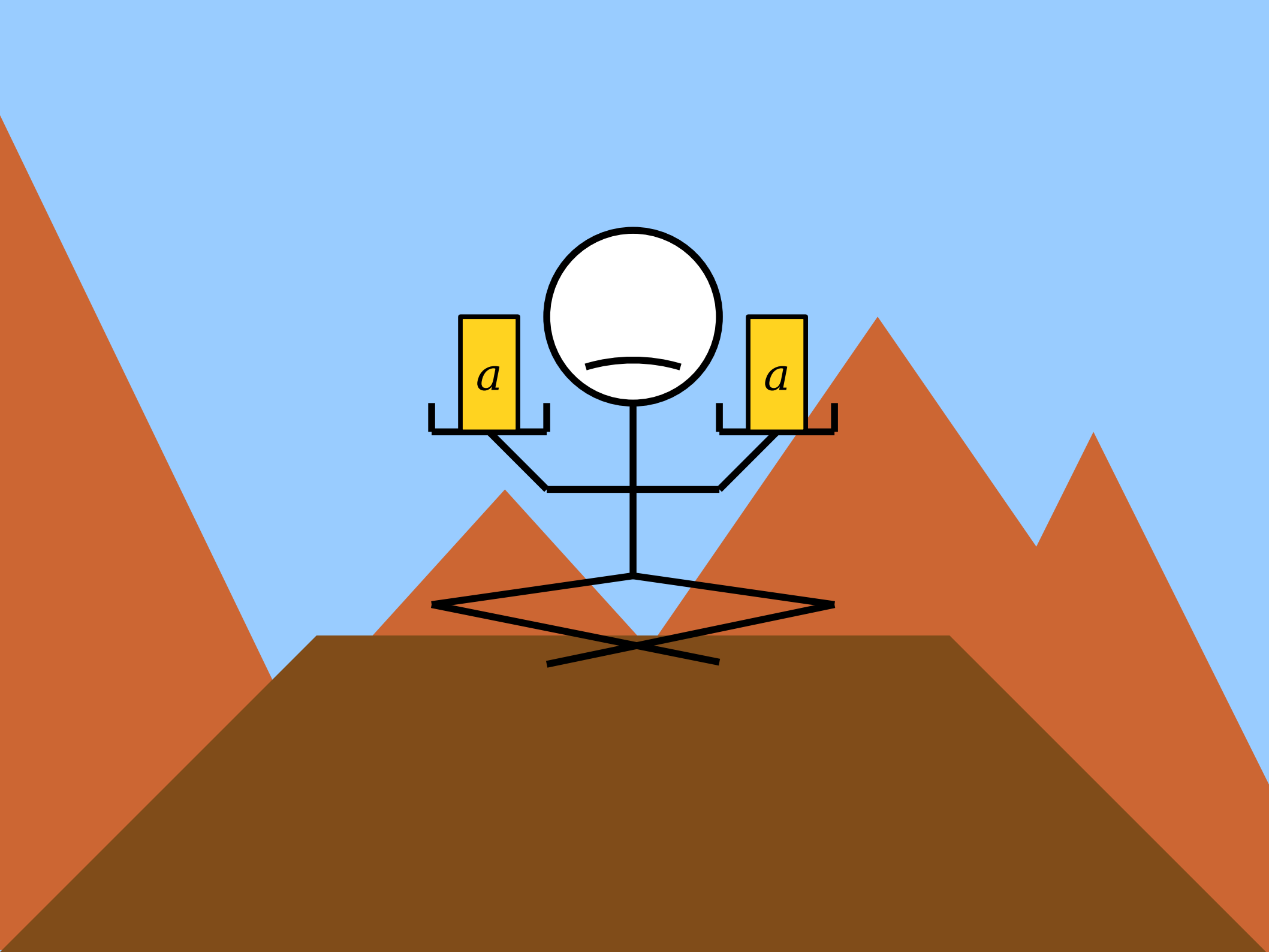
We can think about a binary relation R where aRb means

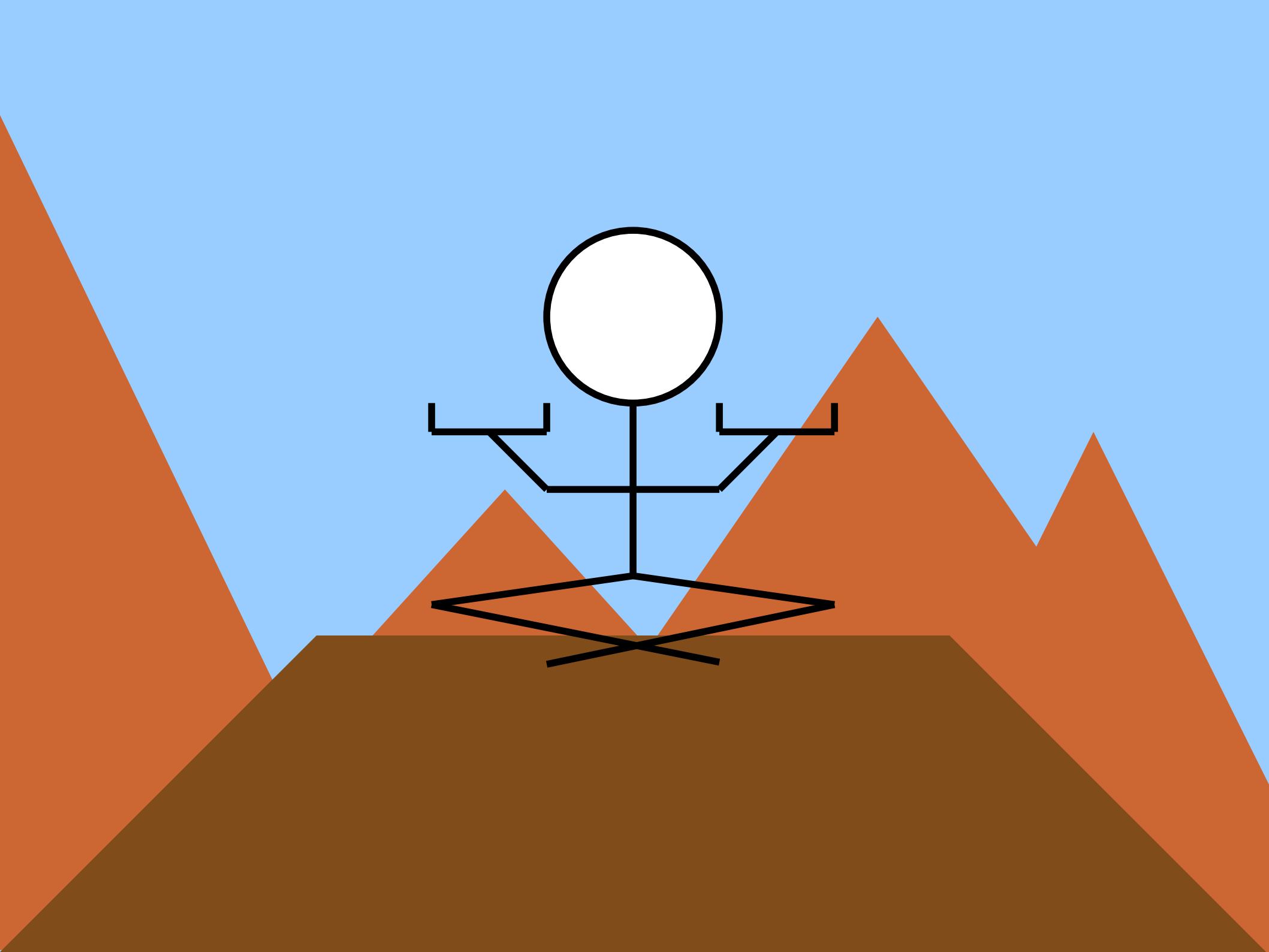
“ a must happen before b ”

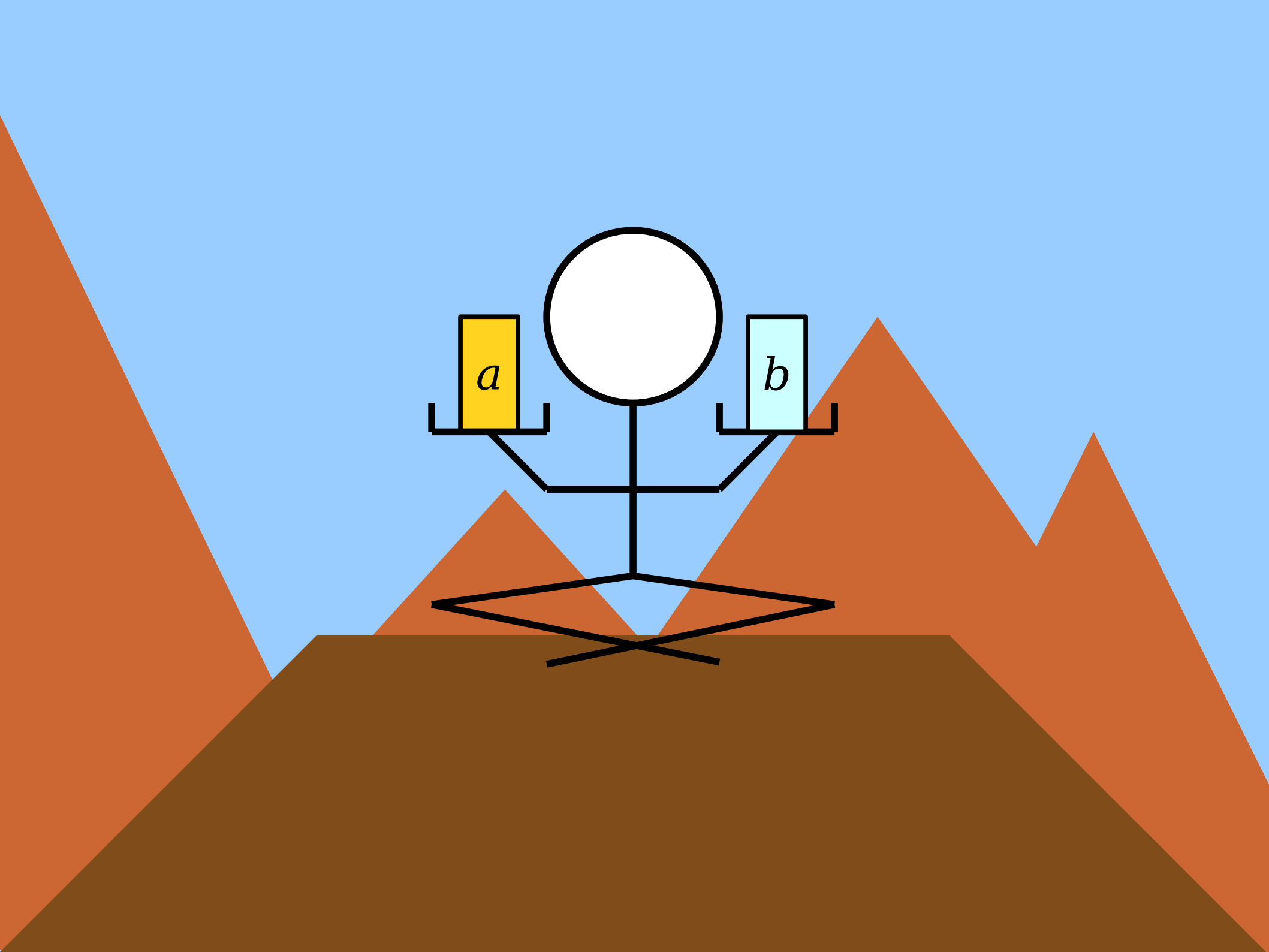
What properties of R could we deduce just from this?

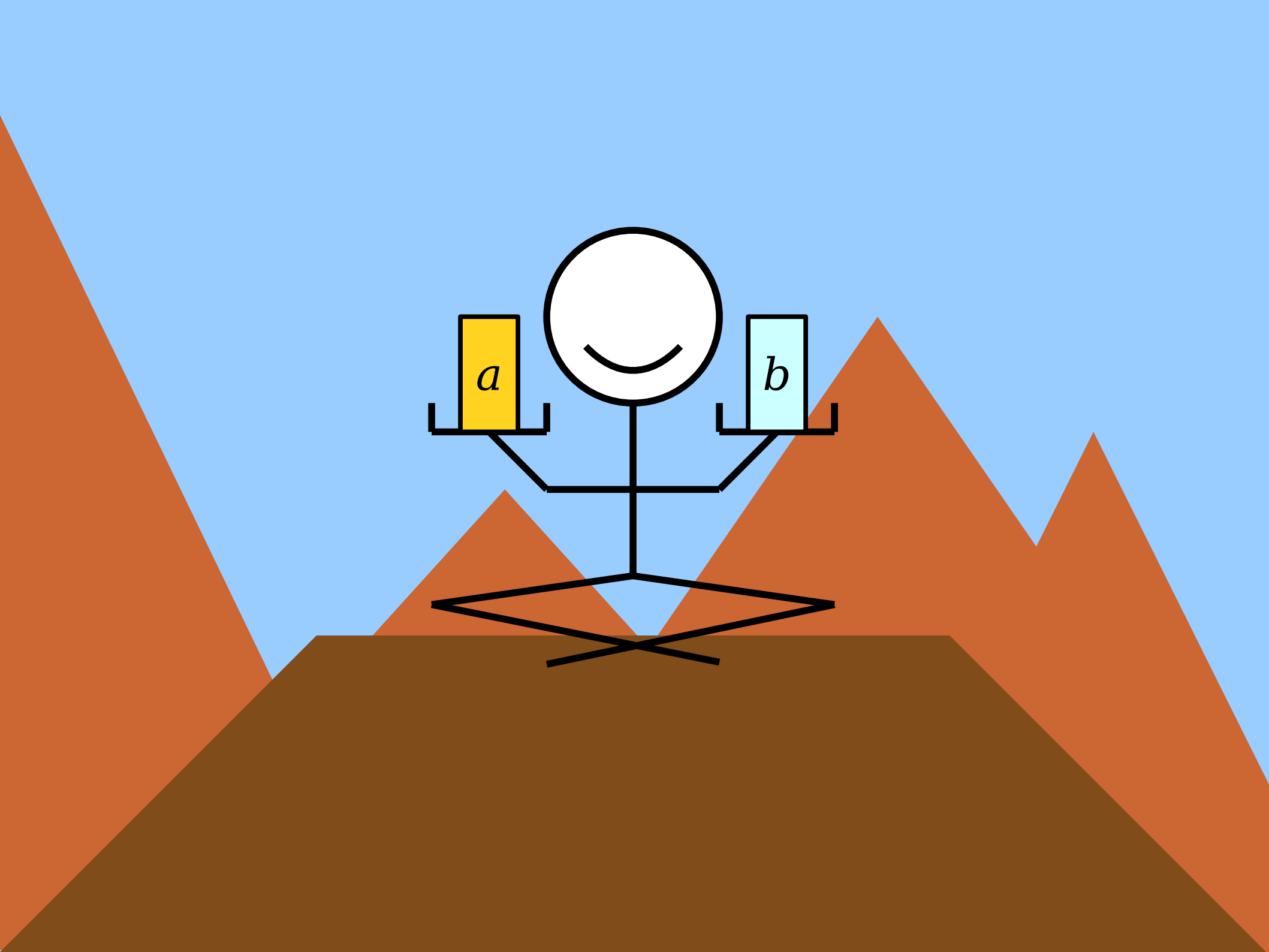


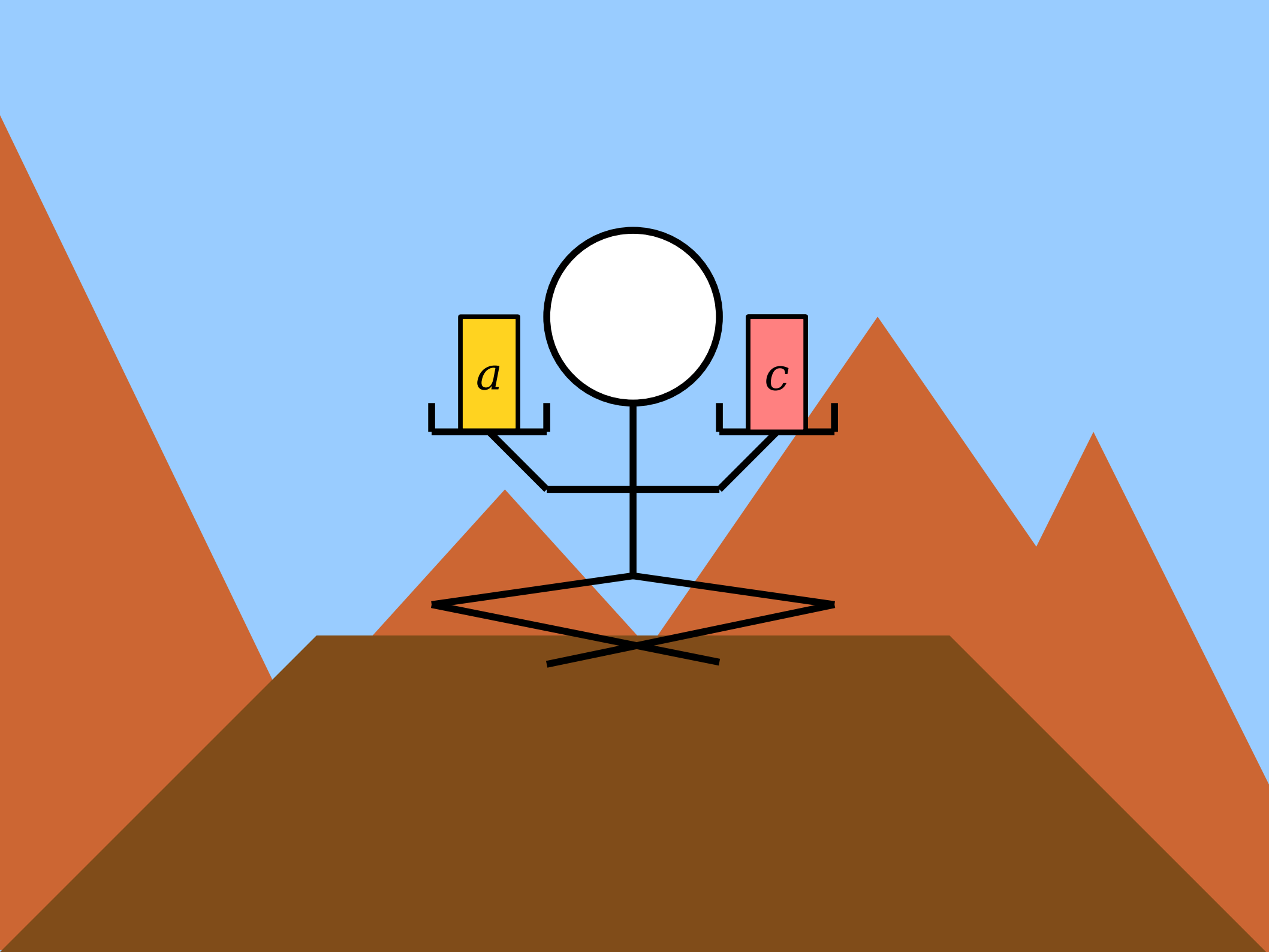


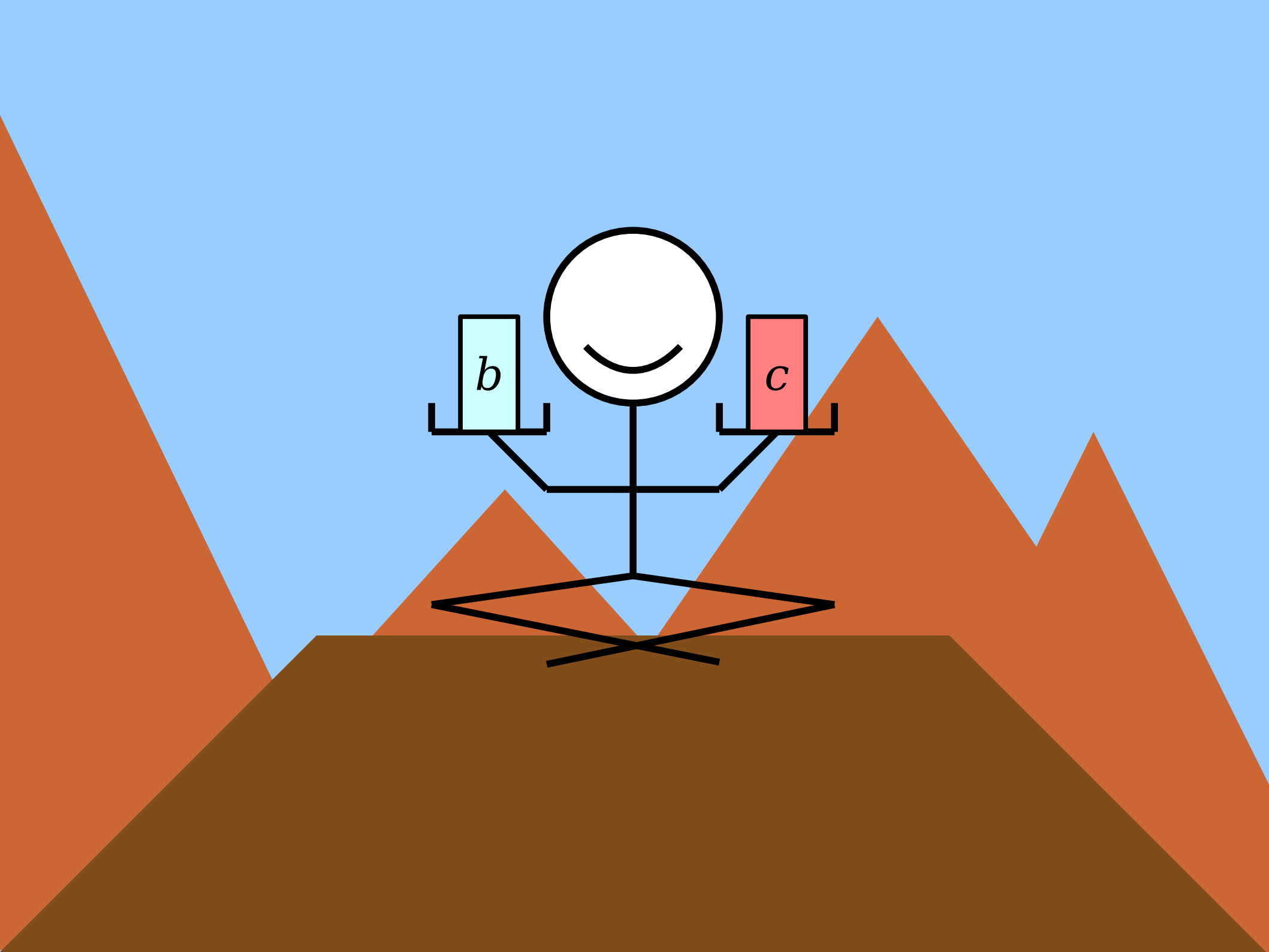


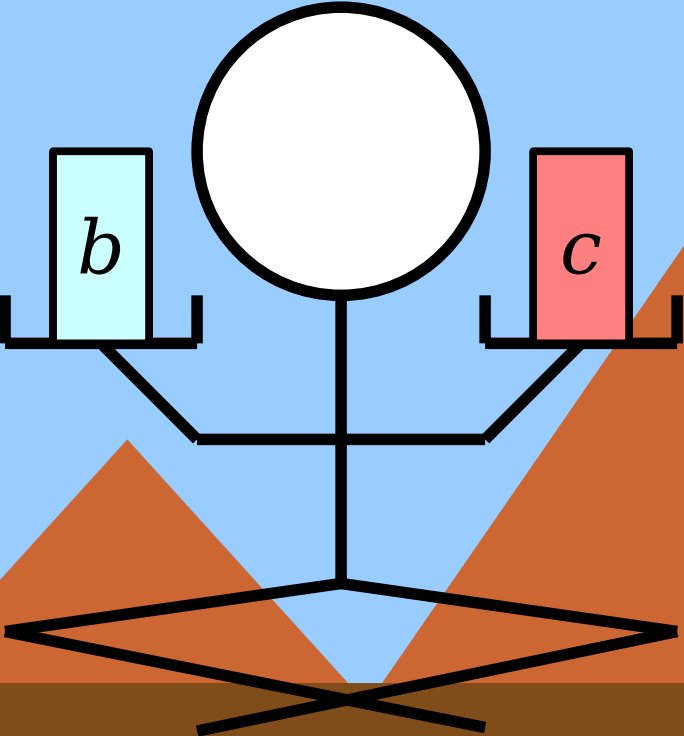
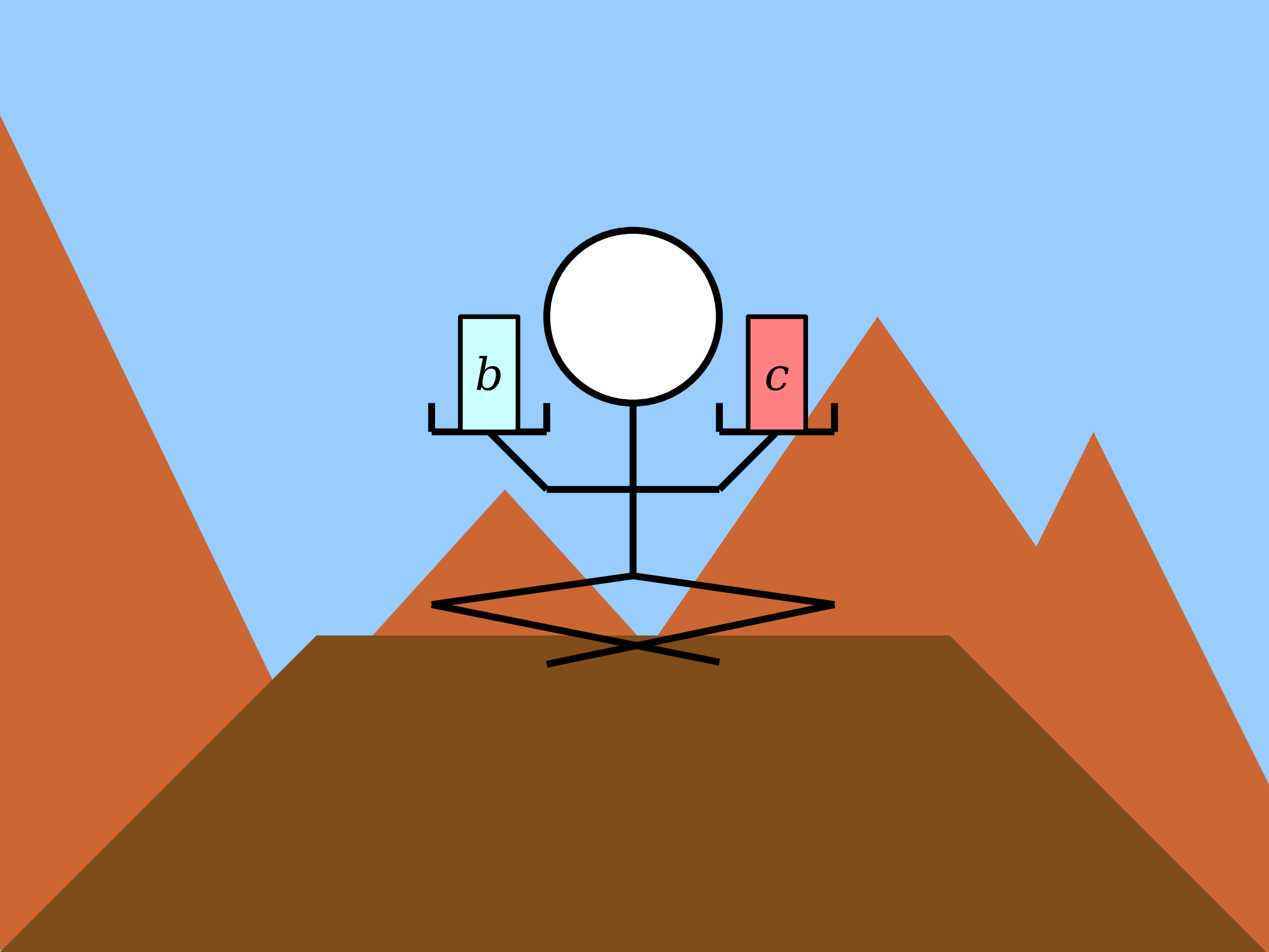


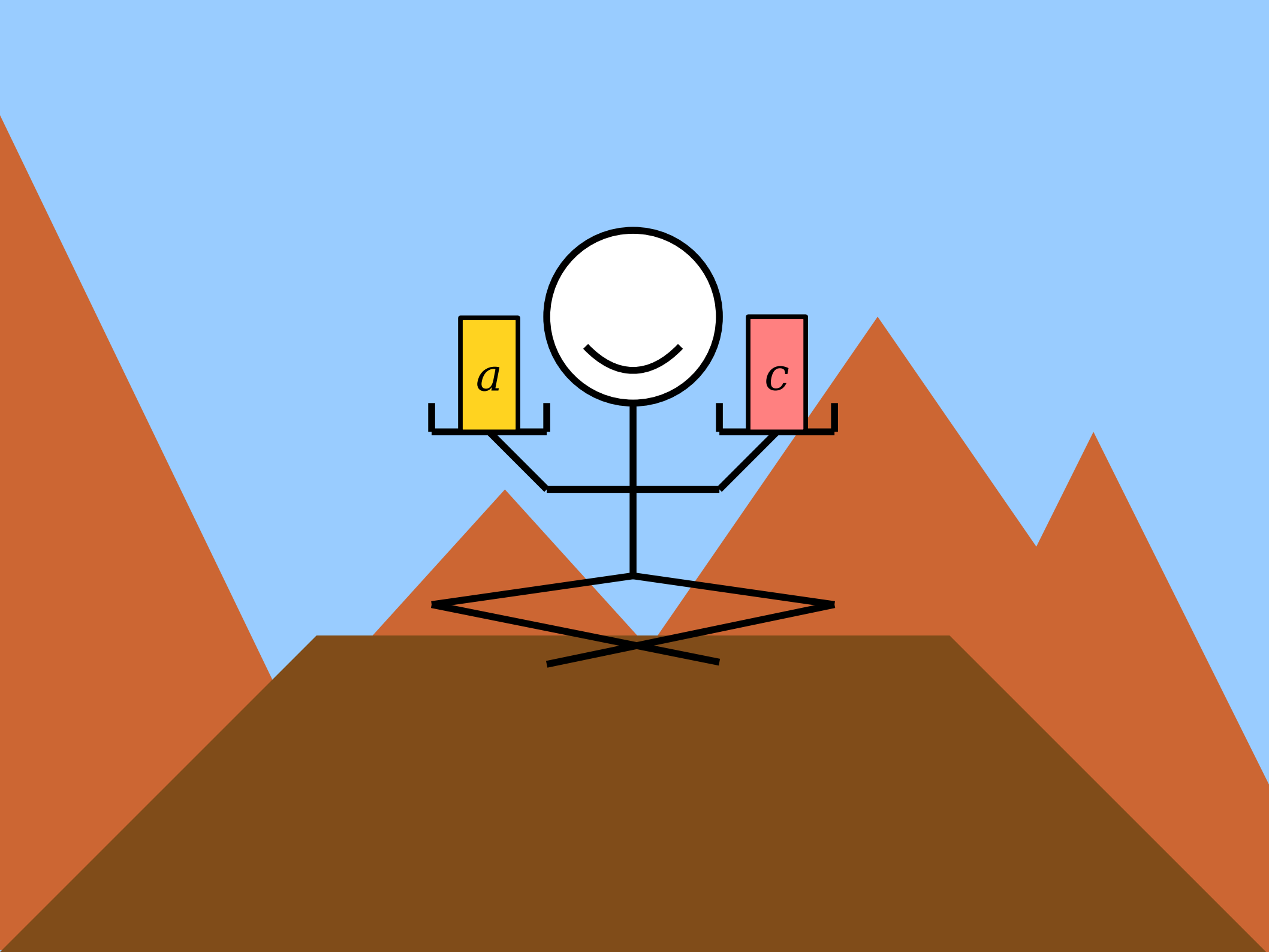






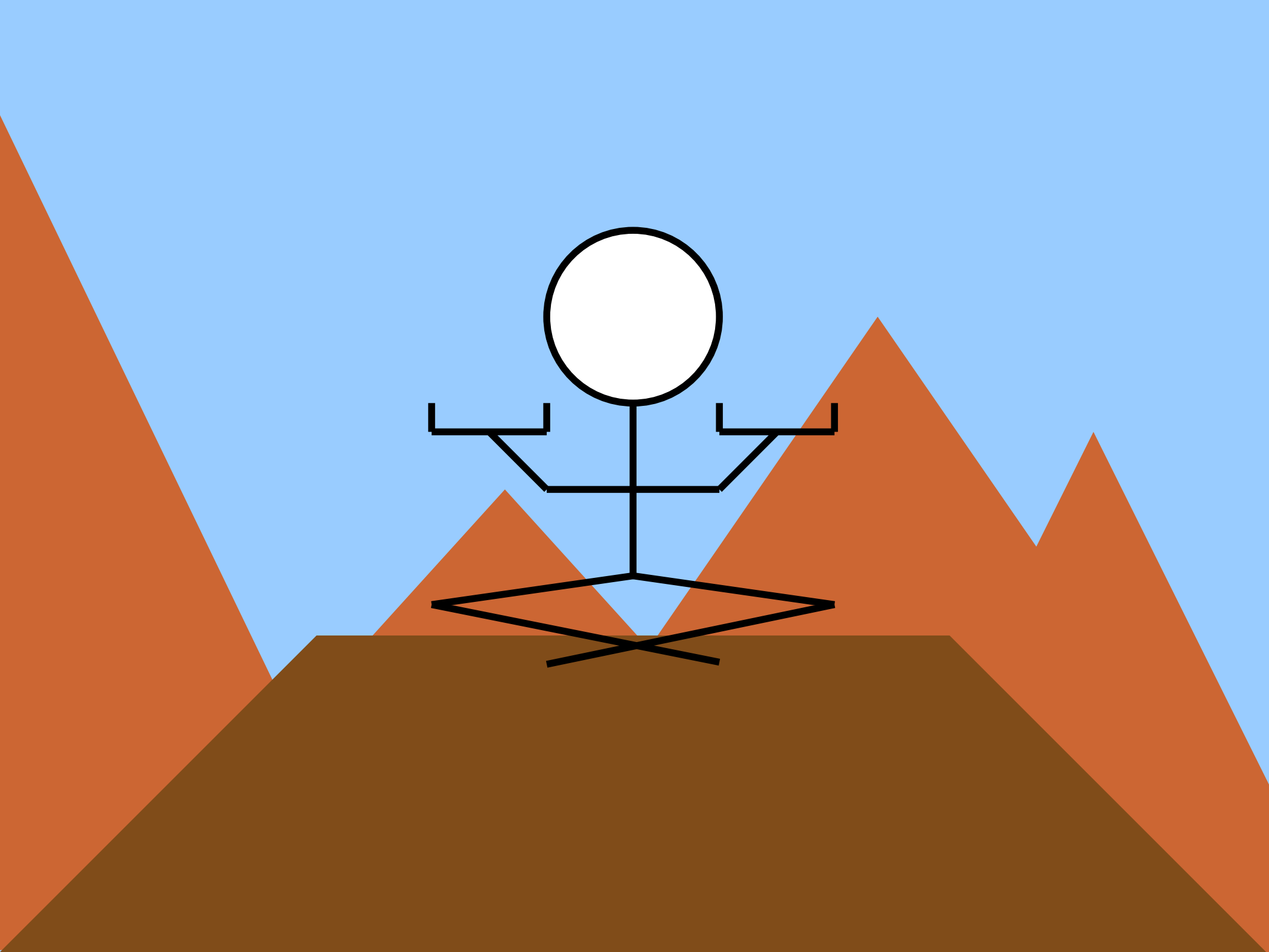


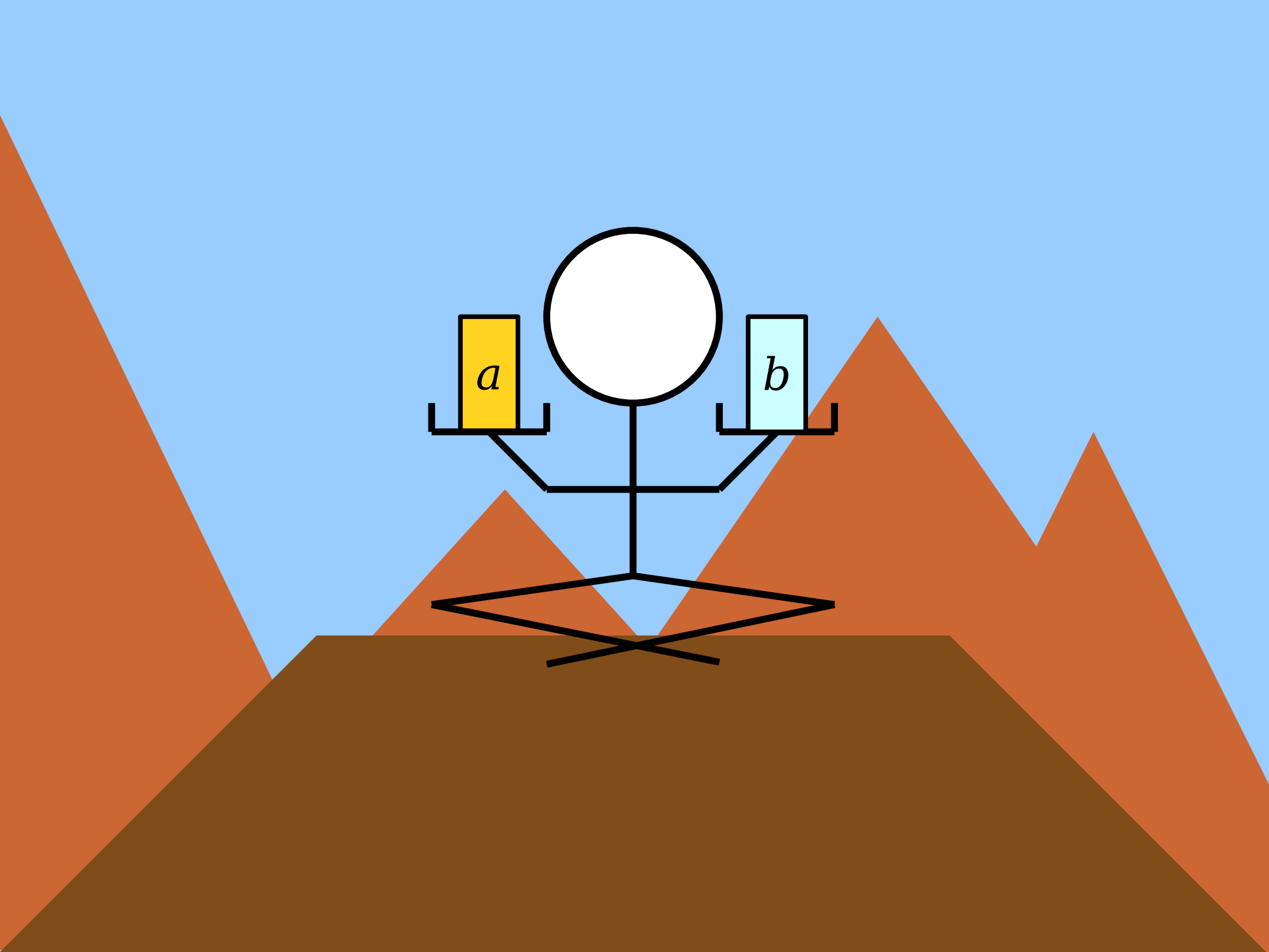


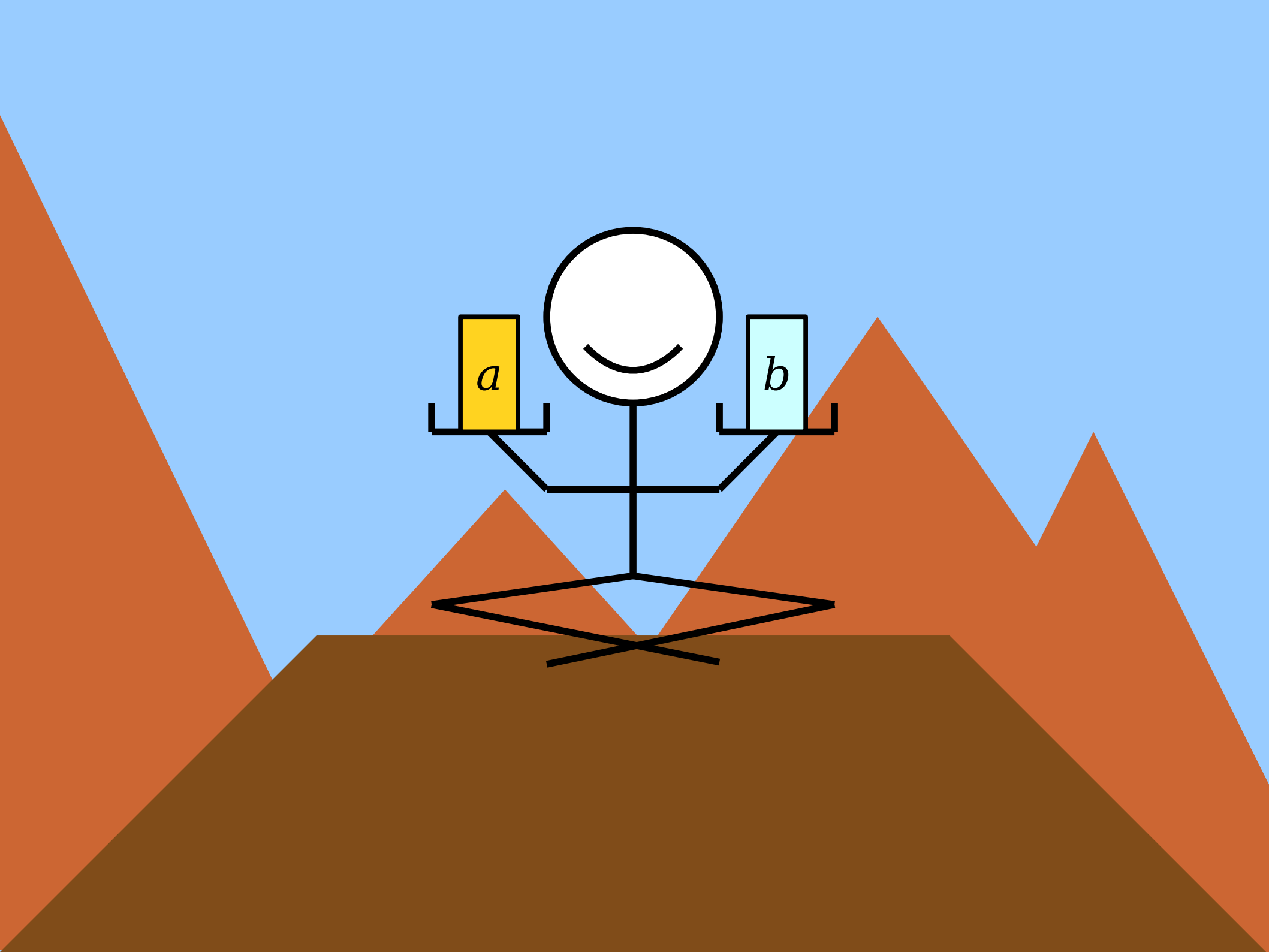


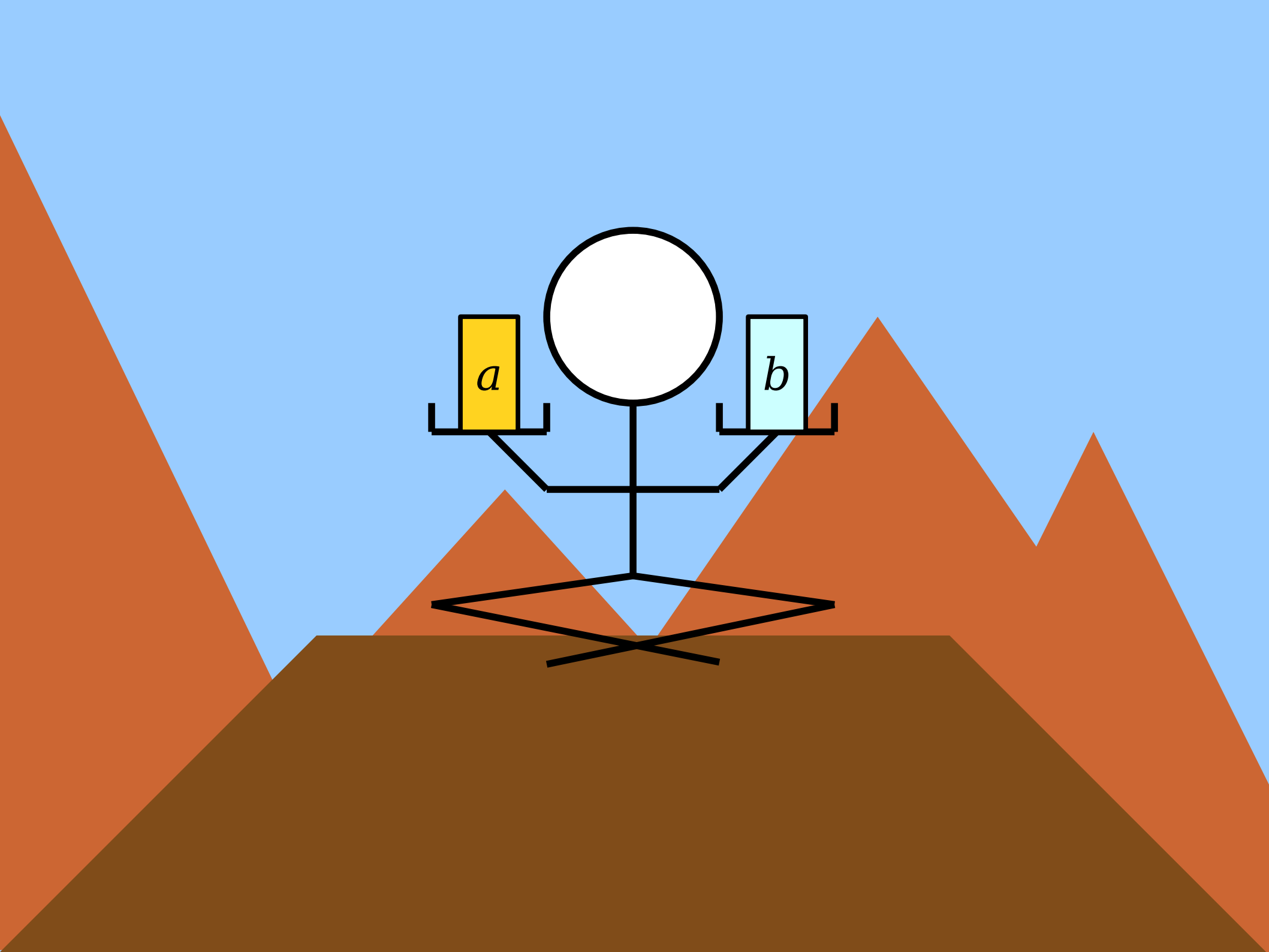
a

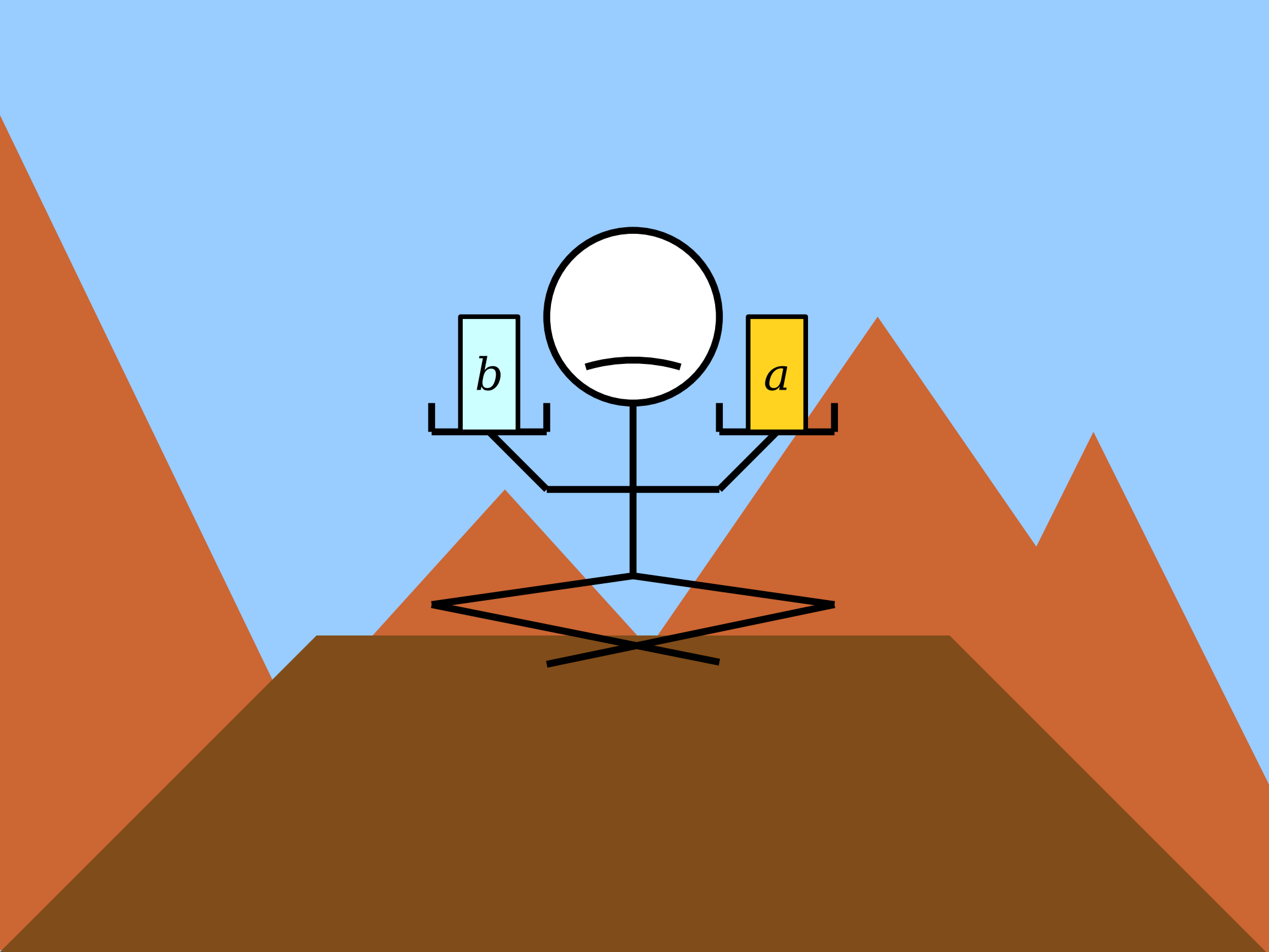
c

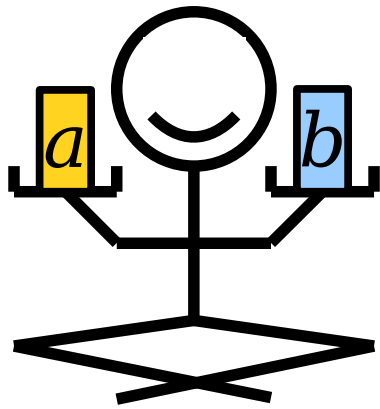
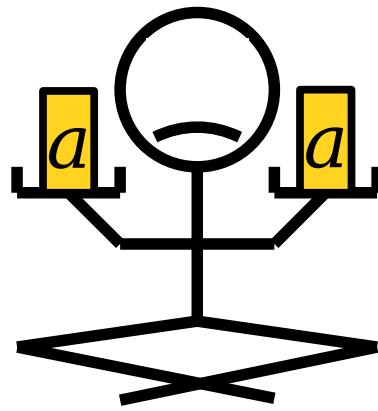




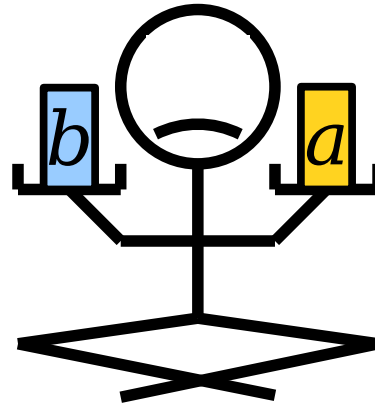
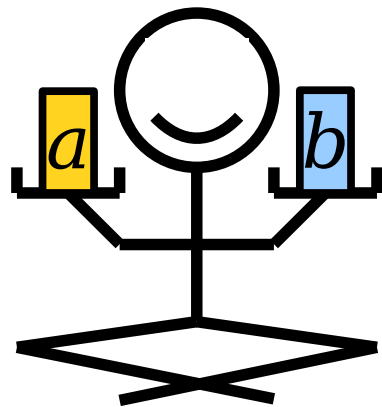
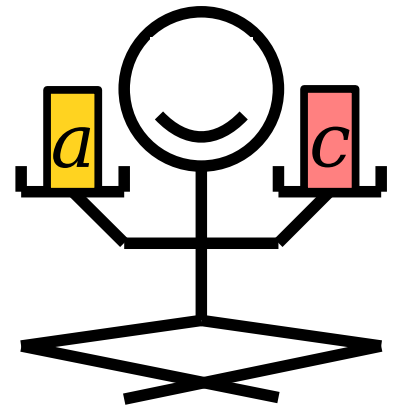
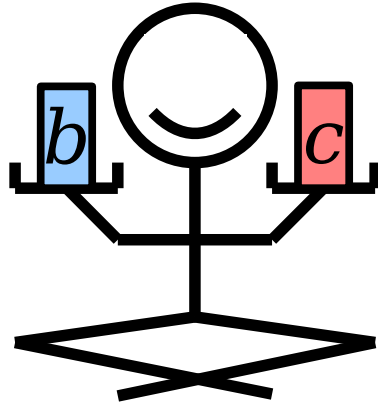








\wedge



$$a \not R a$$

$$a R b \wedge b R c \rightarrow a R c$$

$$a R b \rightarrow b \not R a$$

$$\forall a \in A. a \not R a$$

$$\forall a \in A. \forall b \in A. \forall c \in A. (a R b \wedge b R c \rightarrow a R c)$$

$$\forall a \in A. \forall b \in A. (a R b \rightarrow b \not R a)$$

$$\forall a \in A. a \not R a$$

Transitivity

$$\forall a \in A. \forall b \in A. (a R b \rightarrow b \not R a)$$

$$\forall a \in A. a \not R a$$

Transitivity

$$\forall a \in A. \forall b \in A. (a R b \rightarrow b \not R a)$$

Irreflexivity

Some relations *never* hold from any element to itself.

As an example, $x \not\prec x$ for any x .

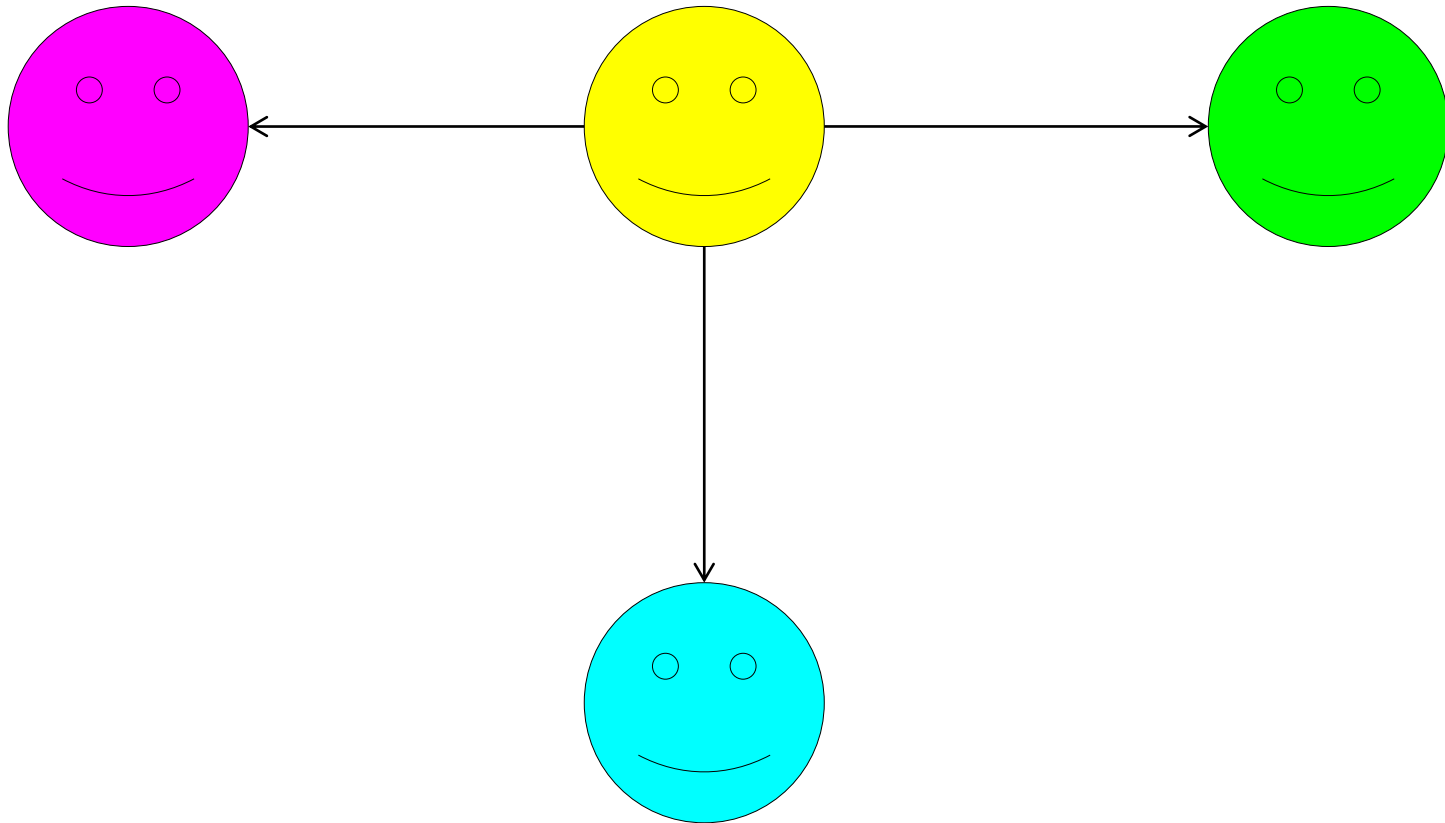
Relations of this sort are called ***irreflexive***.

Formally speaking, a binary relation R over a set A is irreflexive if the following first-order logic statement is true about R :

$$\forall a \in A. a \not R a$$

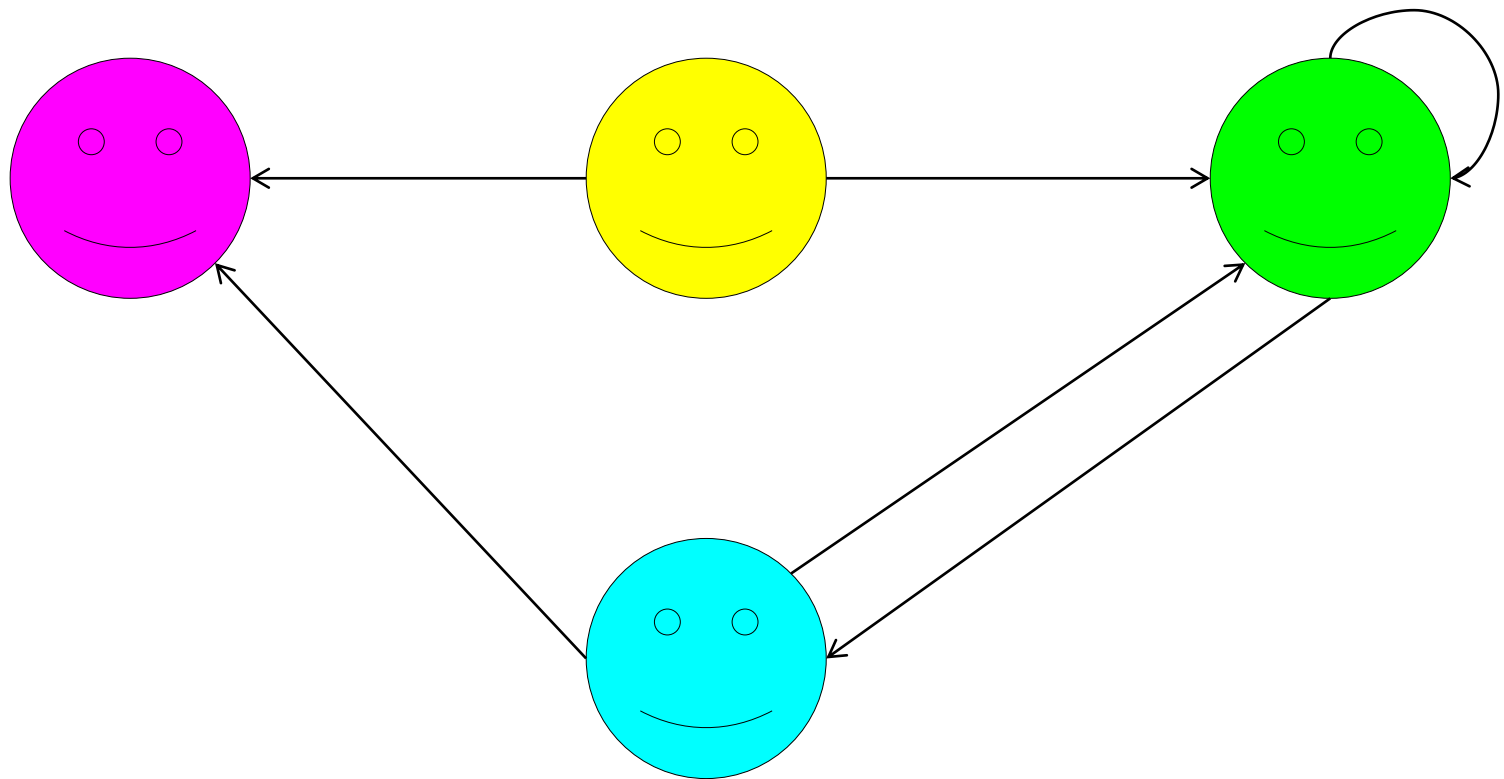
(“*No element is related to itself.*”)

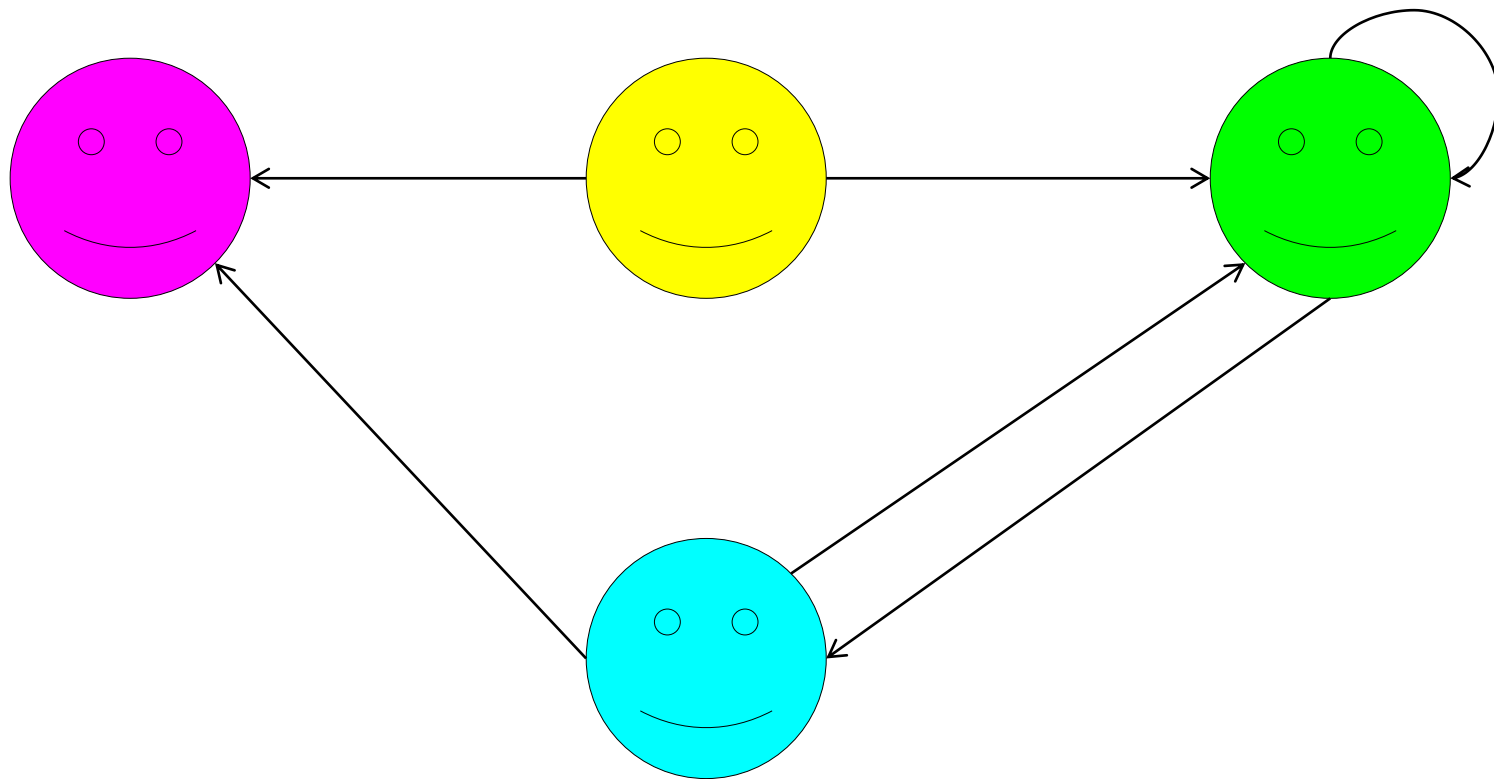
Irreflexivity Visualized



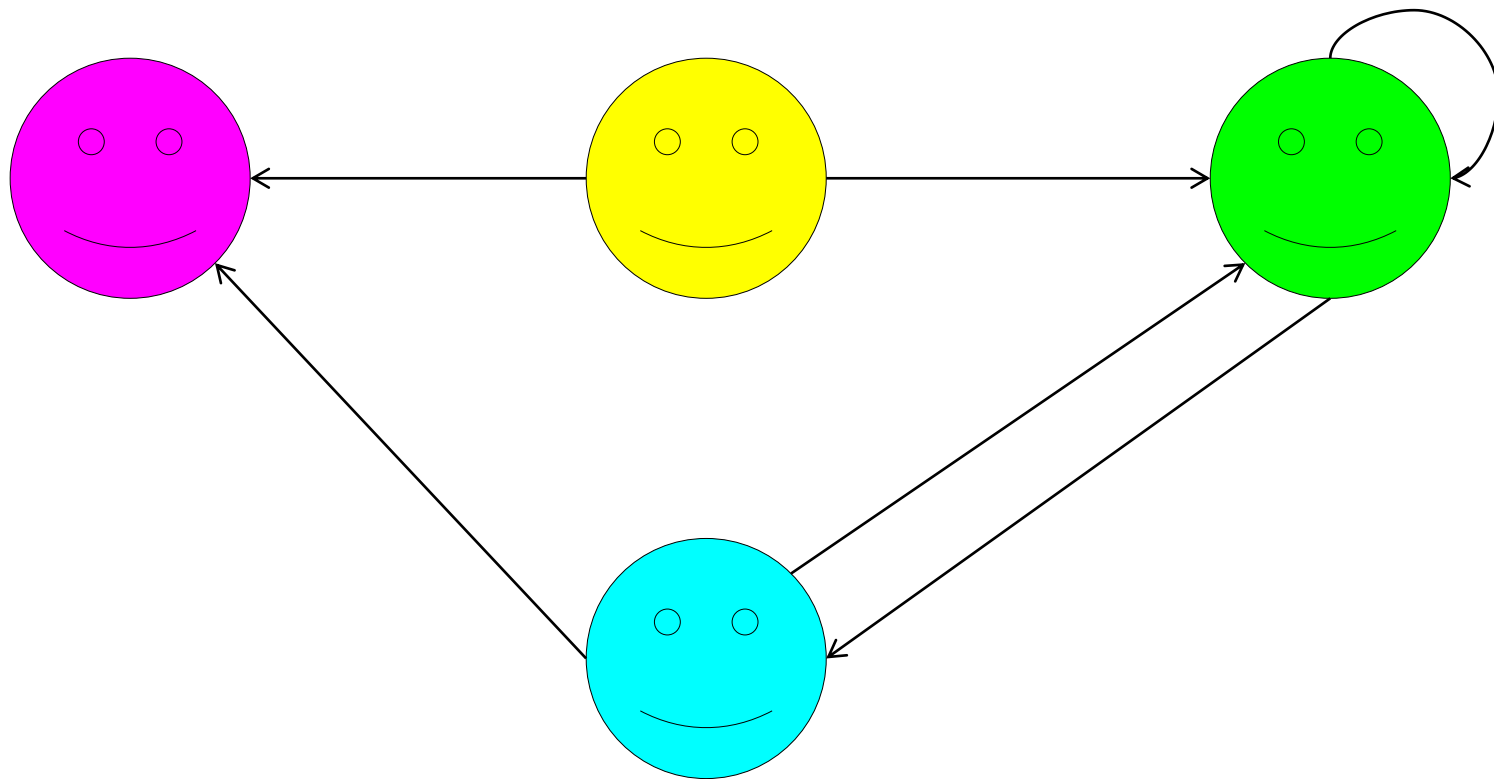
$$\forall a \in A. a \not R a$$

(“No element is related to itself.”)



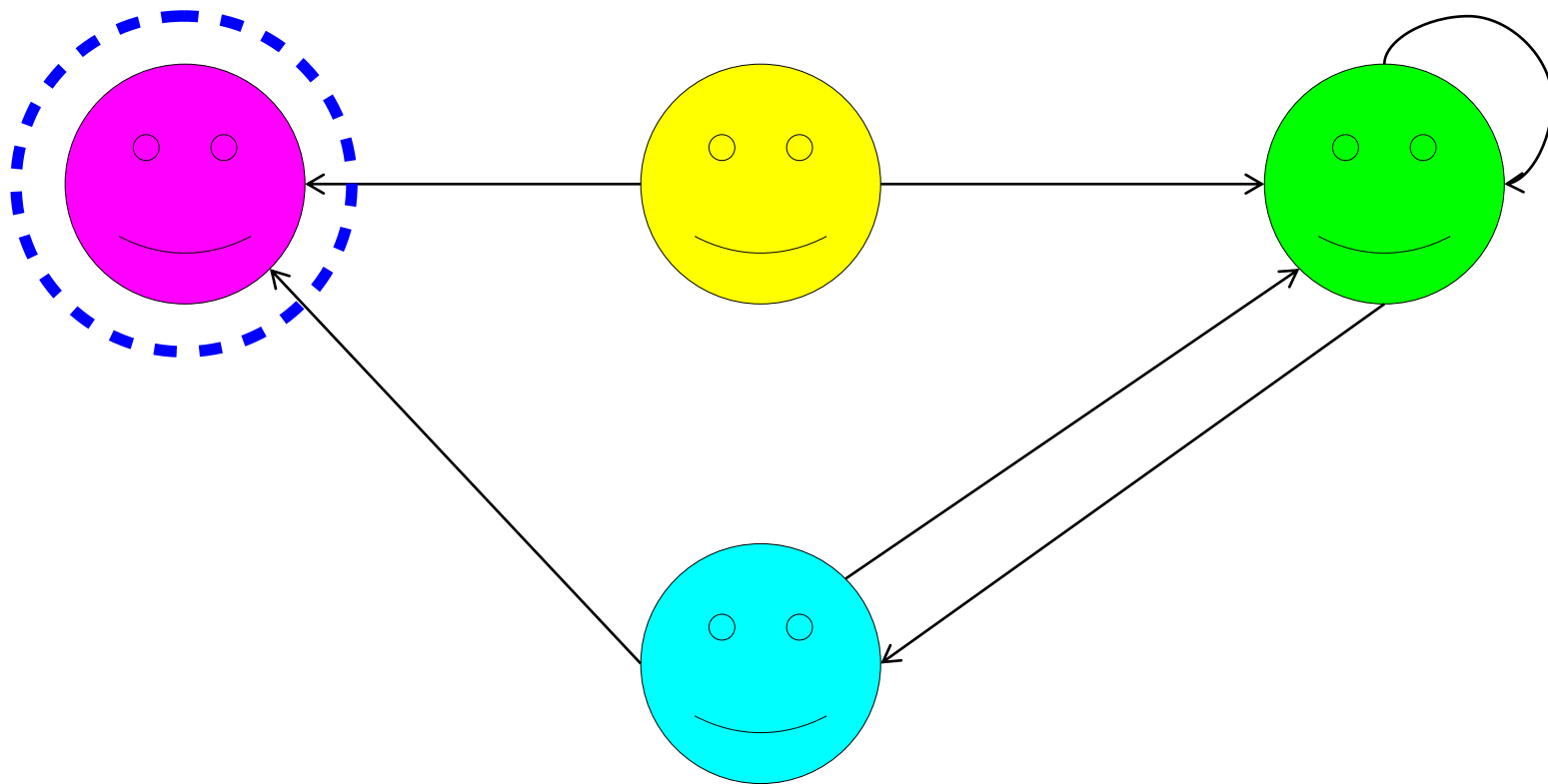


Is this relation reflexive?



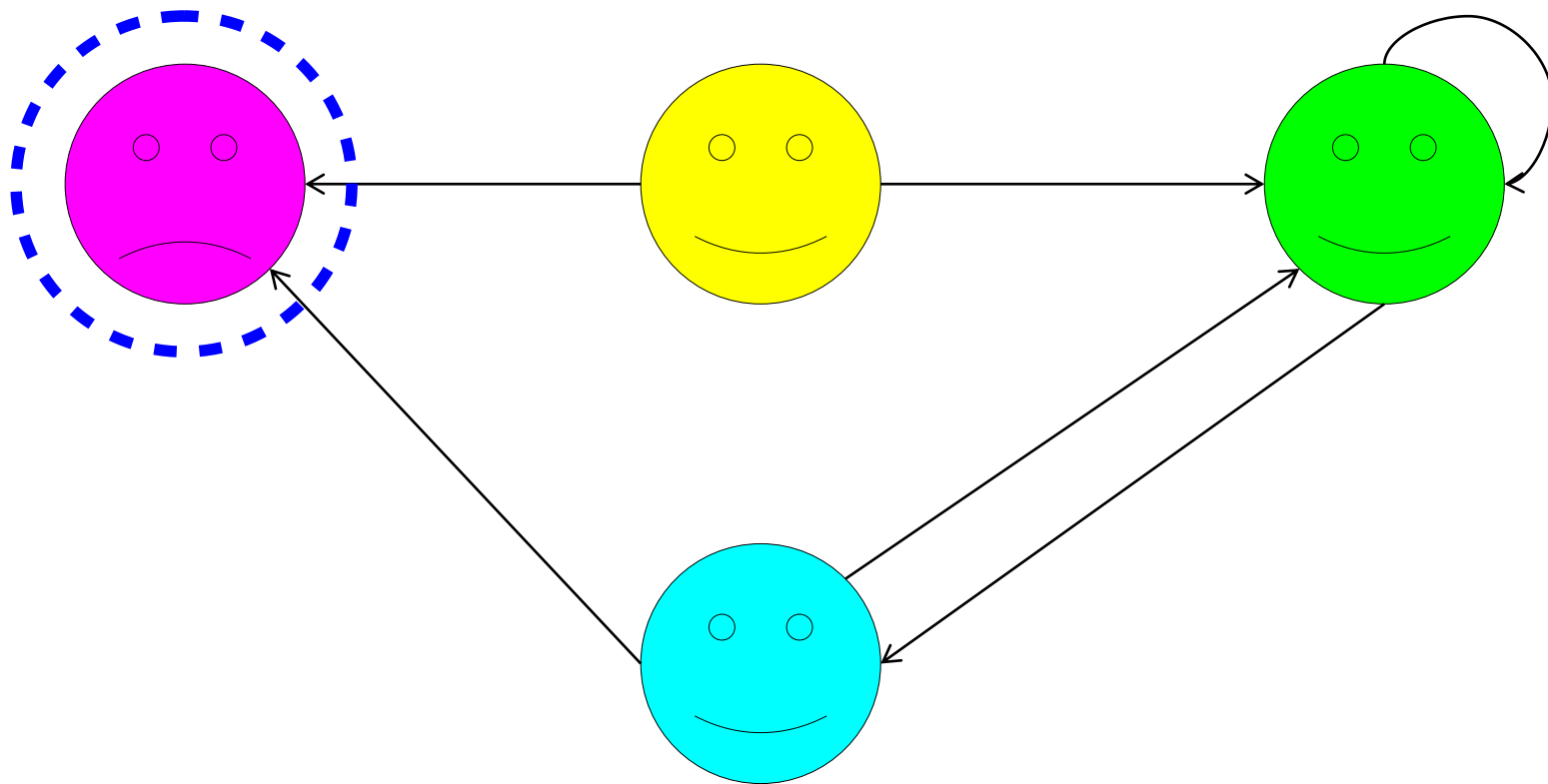
Is this relation reflexive?

$\forall a \in A. aRa$
(“Every element is related to itself.”)



Is this relation
reflexive?

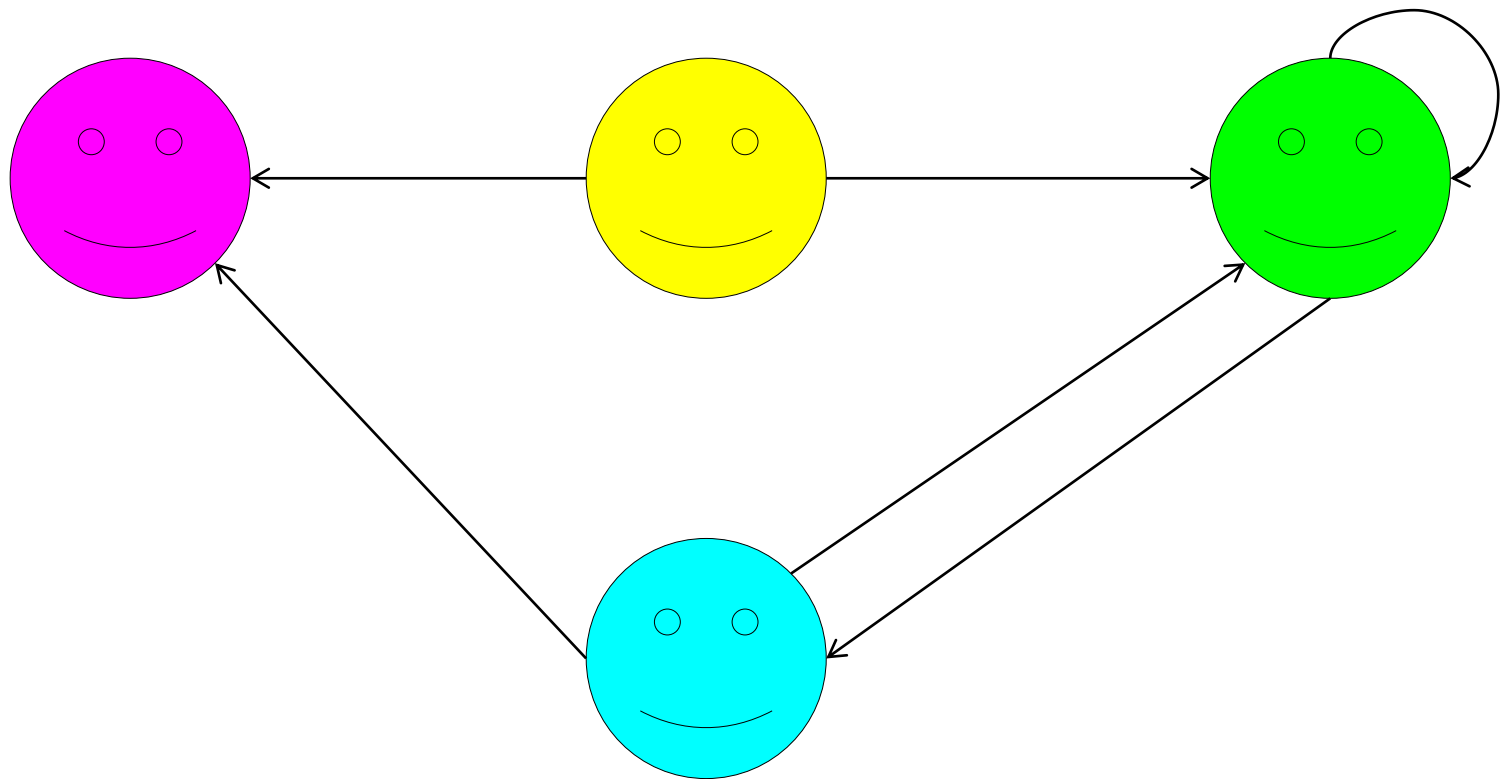
$\forall a \in A. aRa$
(“Every element is related to itself.”)

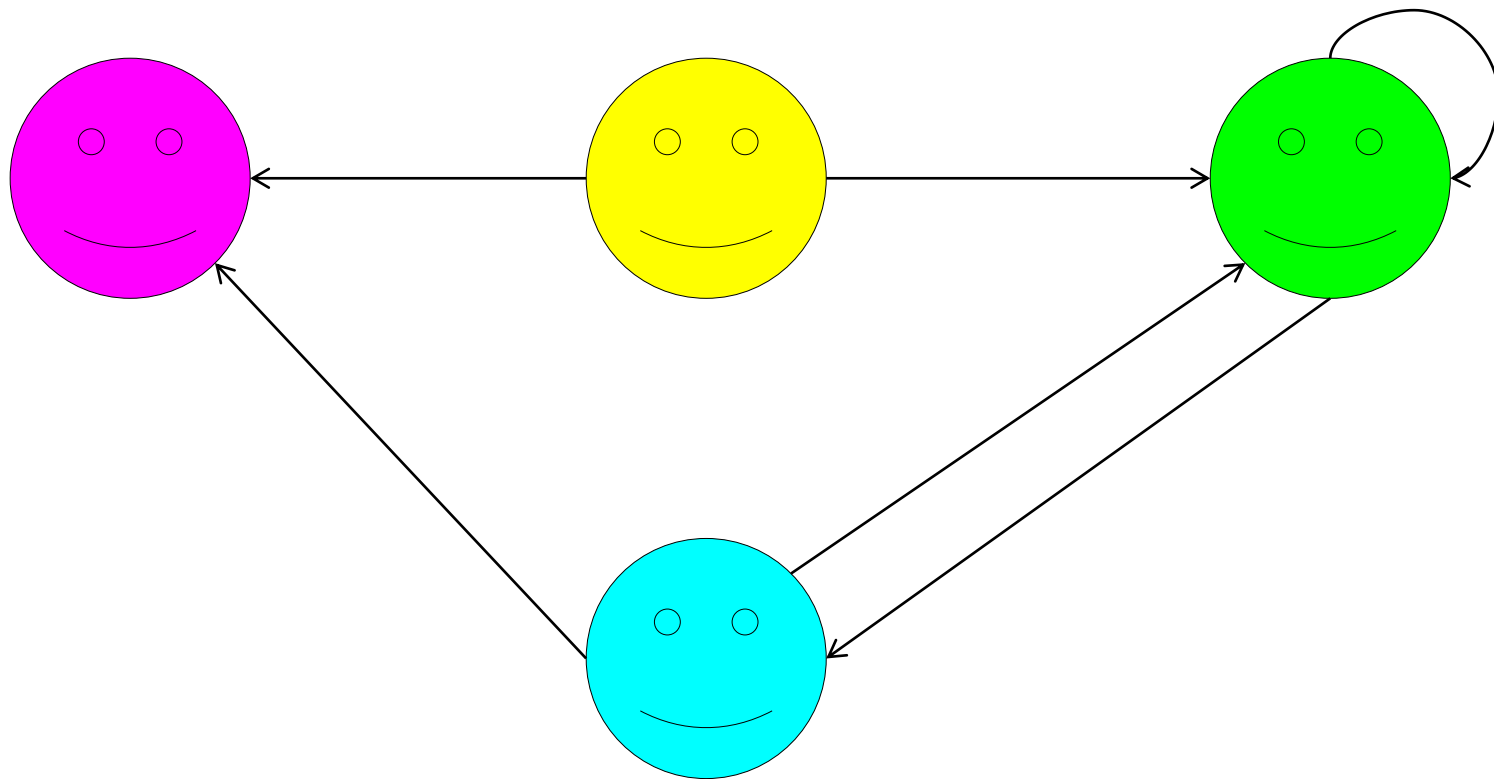


Is this relation
reflexive?

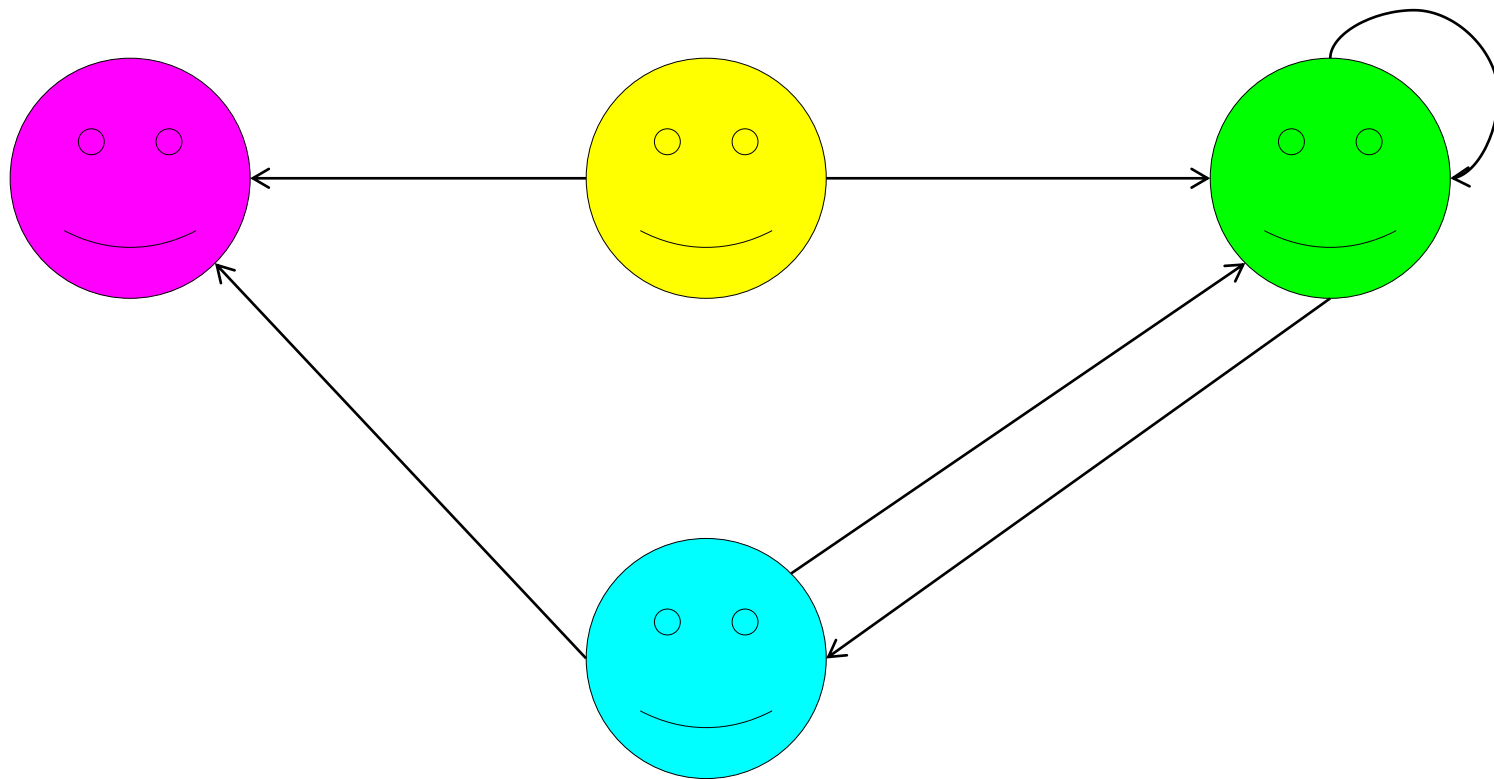
Nope!

$\forall a \in A. aRa$
(“Every element is related to itself.”)



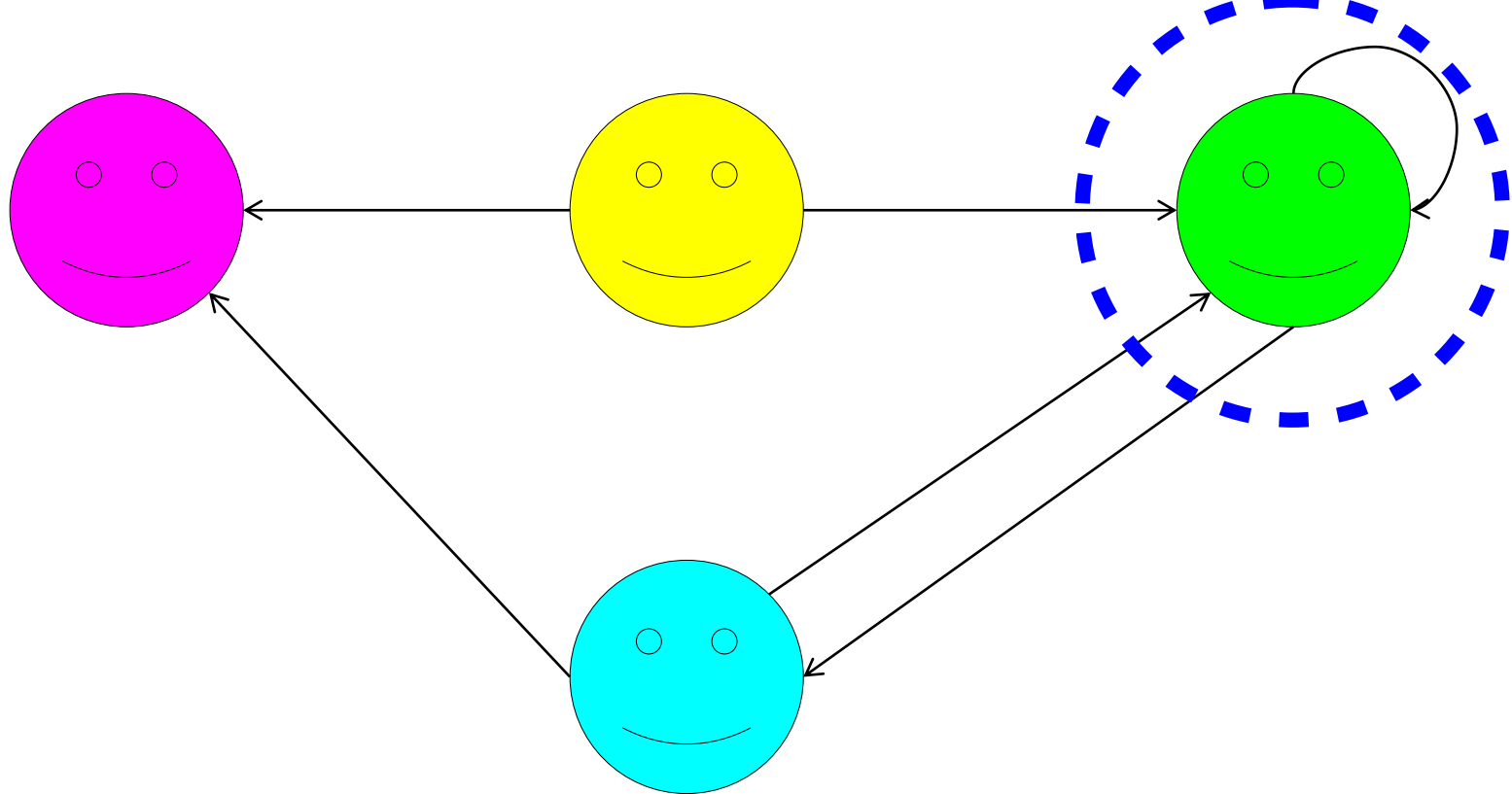


Is this relation
irreflexive?



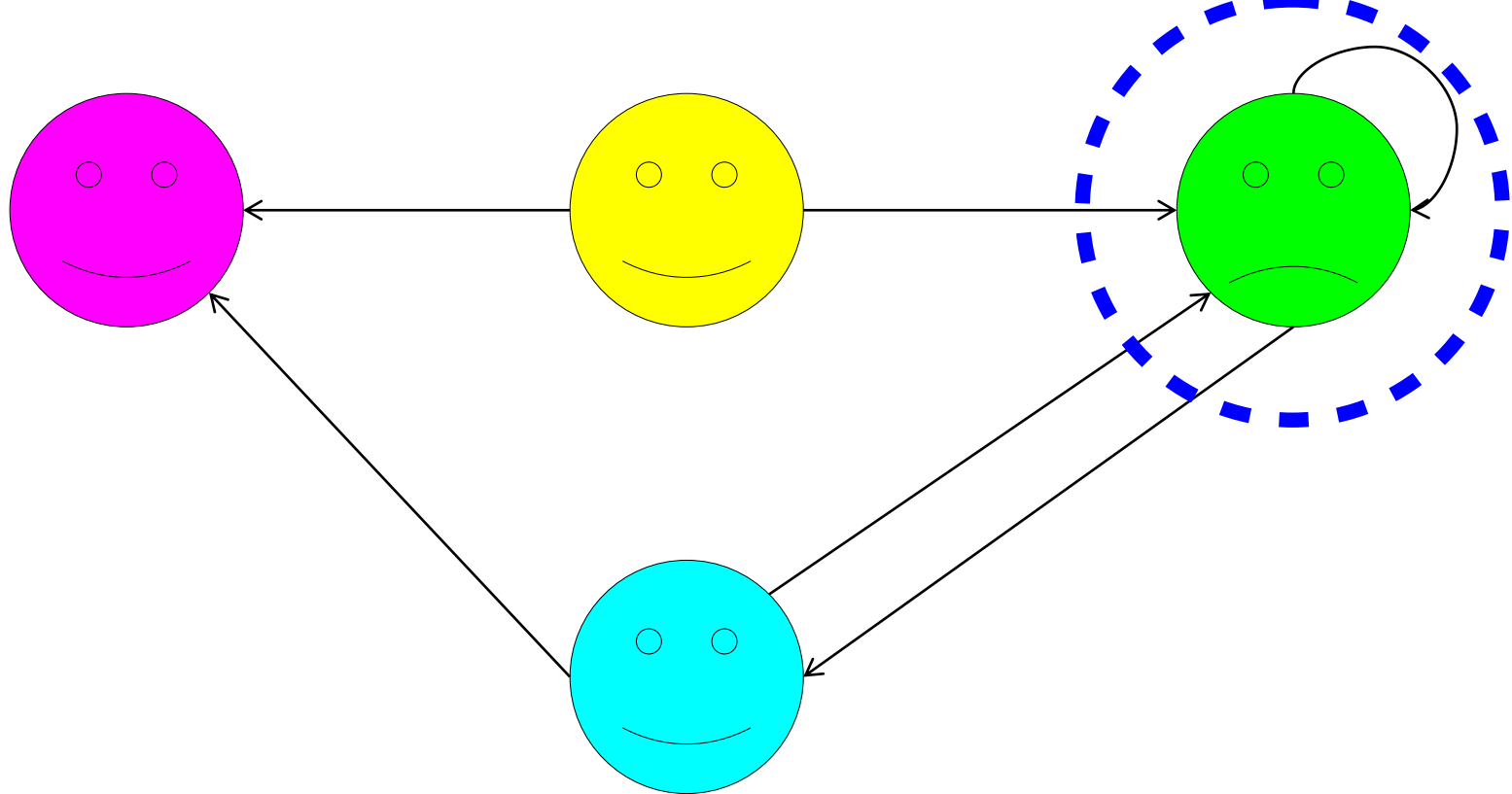
Is this relation
irreflexive?

$\forall a \in A. a \nrightarrow a$
(“No element is related to itself.”)



Is this relation
irreflexive?

$\forall a \in A. a \not R a$
(“No element is related to itself.”)



Is this relation
irreflexive?

Nope!

$\forall a \in A. a \not R a$
("No element is related to itself.")

Reflexivity and Irreflexivity

Reflexivity and irreflexivity are **not** negations of one another!

Here's the definition of reflexivity:

$$\forall a \in A. aRa$$

What is the negation of the above statement?

$$\exists a \in A. aRa$$

What is the definition of irreflexivity?

$$\forall a \in A. a \not R a$$

$$\forall a \in A. a \not R a$$

Transitivity

$$\forall a \in A. \forall b \in A. (a R b \rightarrow b \not R a)$$

Irreflexivity

Transitivity

$$\forall a \in A. \forall b \in A. (aRb \rightarrow b \not R a)$$

Irreflexivity

Transitivity

$$\forall a \in A. \forall b \in A. (aRb \rightarrow b \not R a)$$

Asymmetry

In some relations, the relative order of the objects can never be reversed.

As an example, if $x < y$, then $y \not< x$.

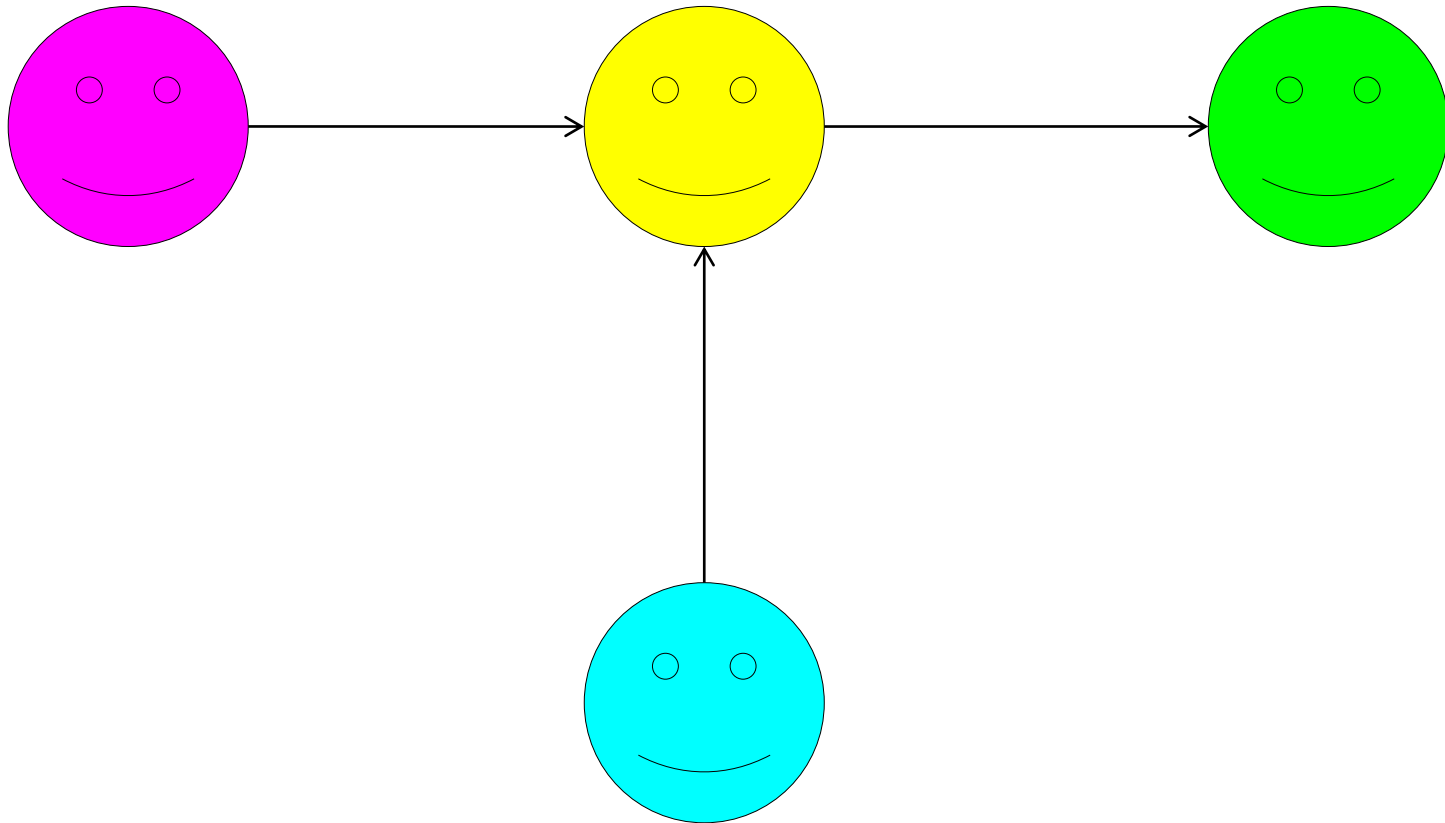
These relations are called ***asymmetric***.

Formally: a binary relation R over a set A is called *asymmetric* if the following first-order logic statement is true about R :

$$\forall a \in A. \forall b \in A. (aRb \rightarrow \neg bRa)$$

(“If a relates to b , then b does not relate to a .”)

Asymmetry Visualized



$\forall a \in A. \forall b \in A. (aRb \rightarrow \neg bRa)$

("If a relates to b, then b does not relate to a.")

Question to Ponder: Are symmetry and asymmetry negations of one another?

Irreflexivity

Transitivity

$$\forall a \in A. \forall b \in A. (aRb \rightarrow b \not R a)$$

Irreflexivity

Transitivity

Asymmetry

Strict Orders

A ***strict order*** is a relation that is irreflexive, asymmetric and transitive.

Some examples:

$$x < y.$$

a can run faster than b .

$A \subsetneq B$ (that is, $A \subseteq B$ and $A \neq B$).

Strict orders are useful for

- representing prerequisite structures,
- modeling dependencies,
- listing preferences,
- and so much more!

Strict Orders IRL

- In C++, many STL containers rely on strict orders to define the relative position of elements in terms of precedence of one item over other.
- Eg. the `std::set` which is implemented with a binary search tree.
- If you want to use `std::sort`, you have to provide a comparator function or overload the `<` operator.
- If you overload the `<` operator, C++ requires that the `<` relation be a strict order over the underlying type!

Recap

Binary Relations

- Reasoning about connections between objects.

Equivalence Relations

- Reasoning about clusters.

Strict Orders

- Reasoning about prerequisites.

Next Time

Proofs involving Binary Relations

- Equivalence Relation Proofs
- Alternative Perspectives on Partitions
- Proofs Involving Multiple Relations