Functions

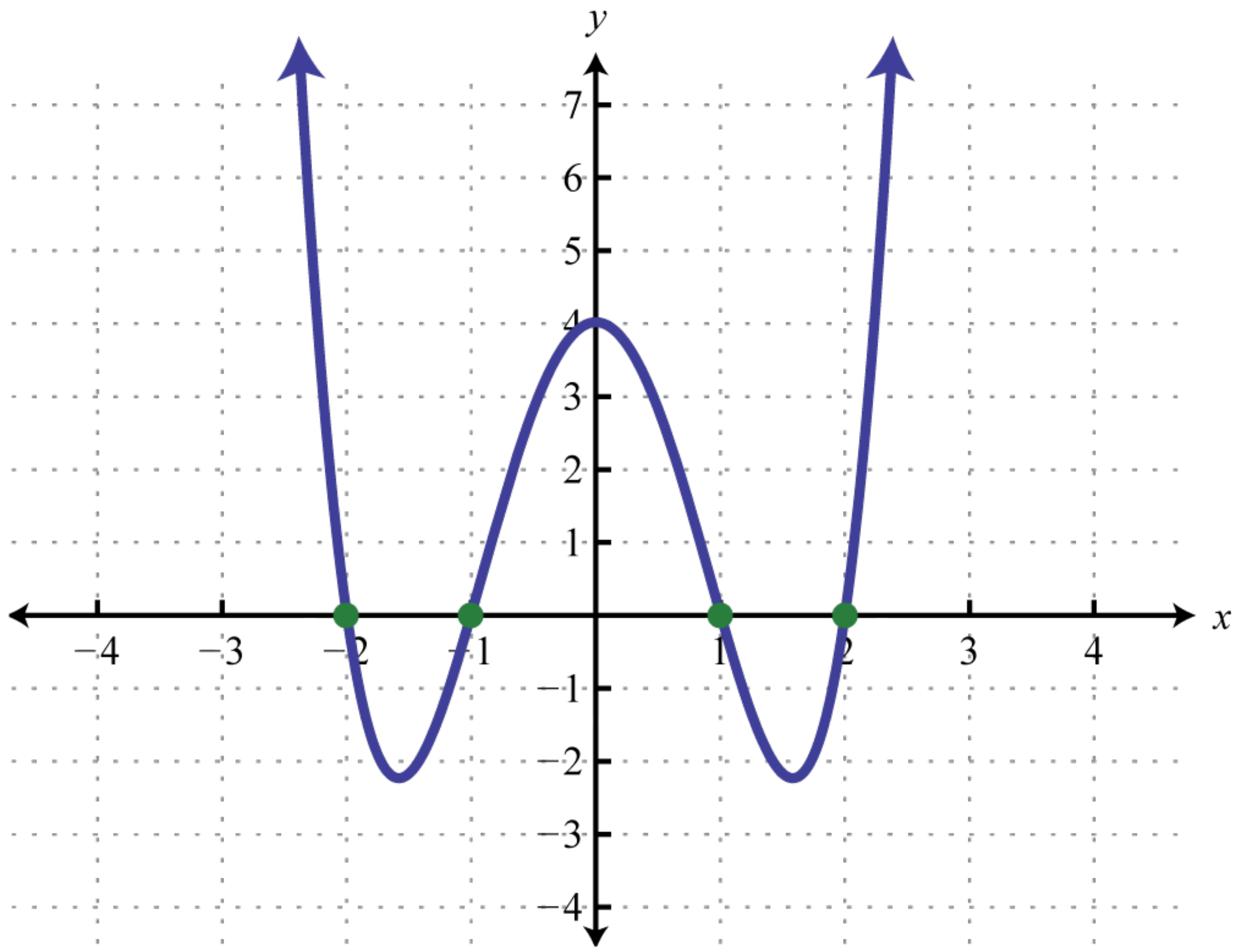
Part One

Outline for Today

- ***What is a Function?***
  - It’s more nuanced than you might expect.
- ***Domains and Codomains***
  - Where functions start, and where functions end.
- ***Defining a Function***
  - Expressing transformations compactly.
- ***Special Classes of Functions***
  - Useful types of functions you’ll encounter IRL.
- ***Proofs on First-Order Definitions***
  - A key skill.

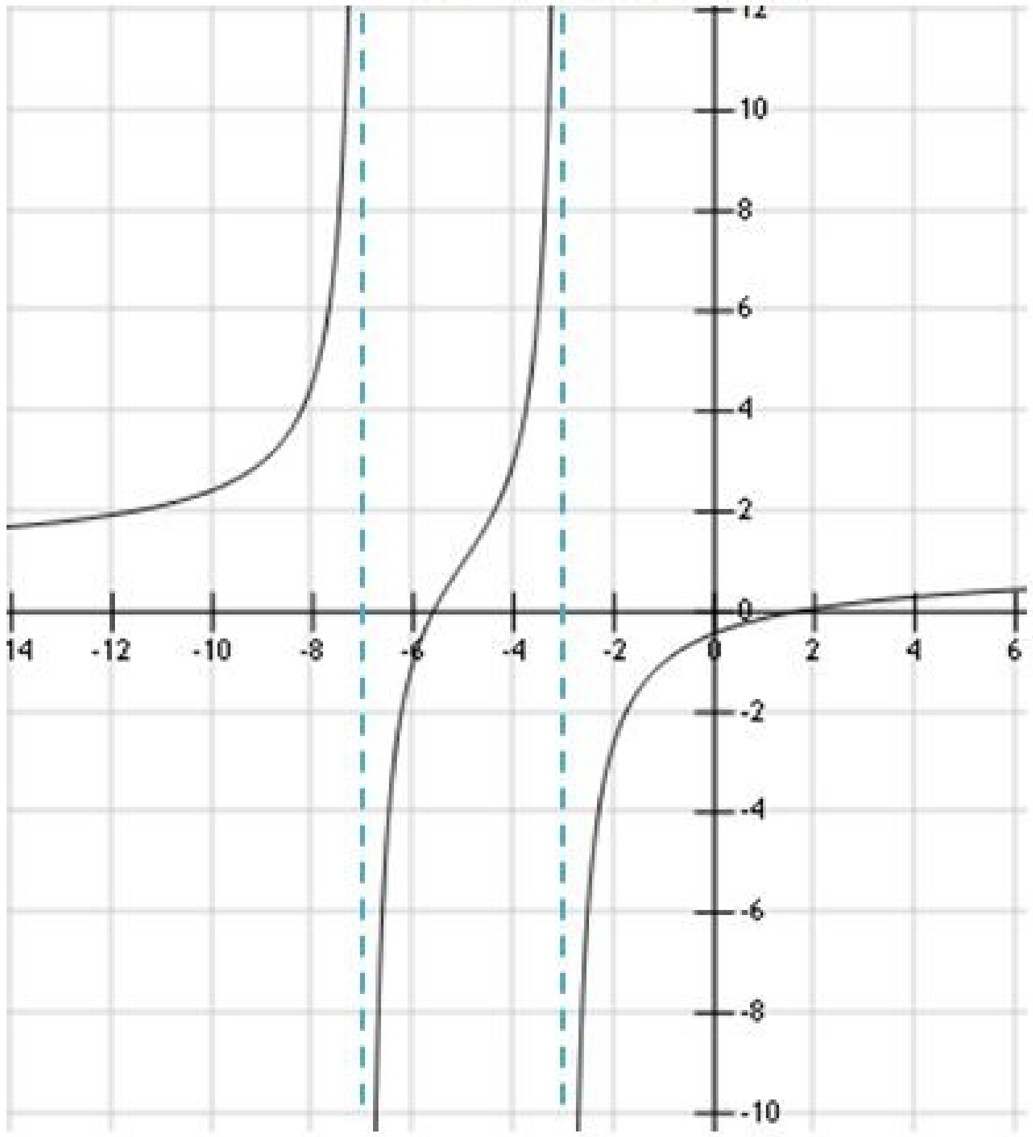
What is a function?

Functions, High-School Edition



$$f(x) = x^4 - 5x^2 + 4$$

$$f(x) = \frac{x^2 + 4x - 9}{x^2 + 10x + 21}$$



source: http://study.com/cimages/multimages/16/asymptote_1.JPG

Functions, High-School Edition

- In high school, functions are usually given as objects of the form

$$f(x) = \frac{x^3+3x^2+15x+7}{1-x^{137}}$$

- What does a function do?
  - It takes in as input a real number.
  - It outputs a real number
  - … except when there are vertical asymptotes or other discontinuities, in which case the function doesn't output anything.

Functions, CS Edition

```
int flipUntil(int n) {
  int numHeads = 0;
  int numTries = 0;

  while (numHeads < n) {
    if (randomBoolean()) {
      numHeads++;
    }
    numTries++;
  }

  return numTries;
}
```

Functions, CS Edition

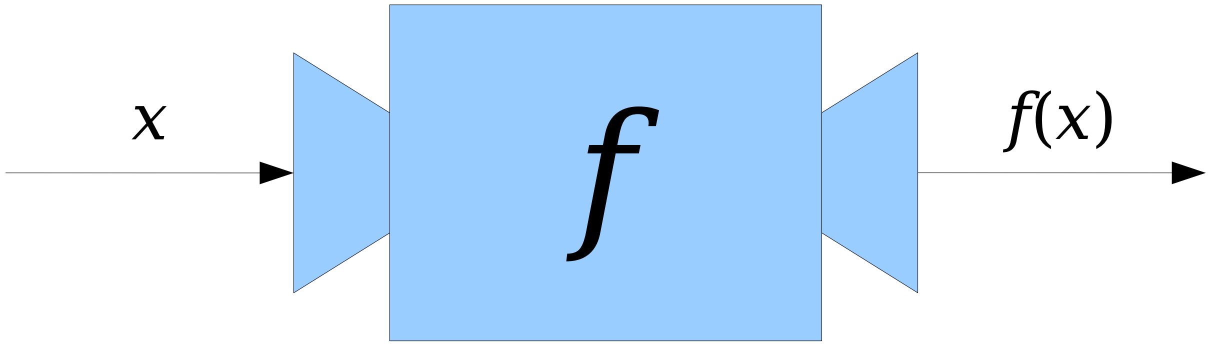
- In programming, functions
  - might take in inputs,
  - might return values,
  - might have side effects,
  - might never return anything,
  - might crash, and
  - might return different values when called multiple times.

What's Common?

- Although high-school math functions and CS functions are pretty different, they have two key aspects in common:
  - They take in inputs.
  - They produce outputs.
- In math, we like to keep things easy, so that's pretty much how we're going to define a function.

High-Level Intuition:

A function is an object f that takes in exactly one input x and produces exactly one output $f(x)$.



(This is not definition. It's just to help you build and intuition.)

High School versus CS Functions

- In high school, functions usually were given by a rule:

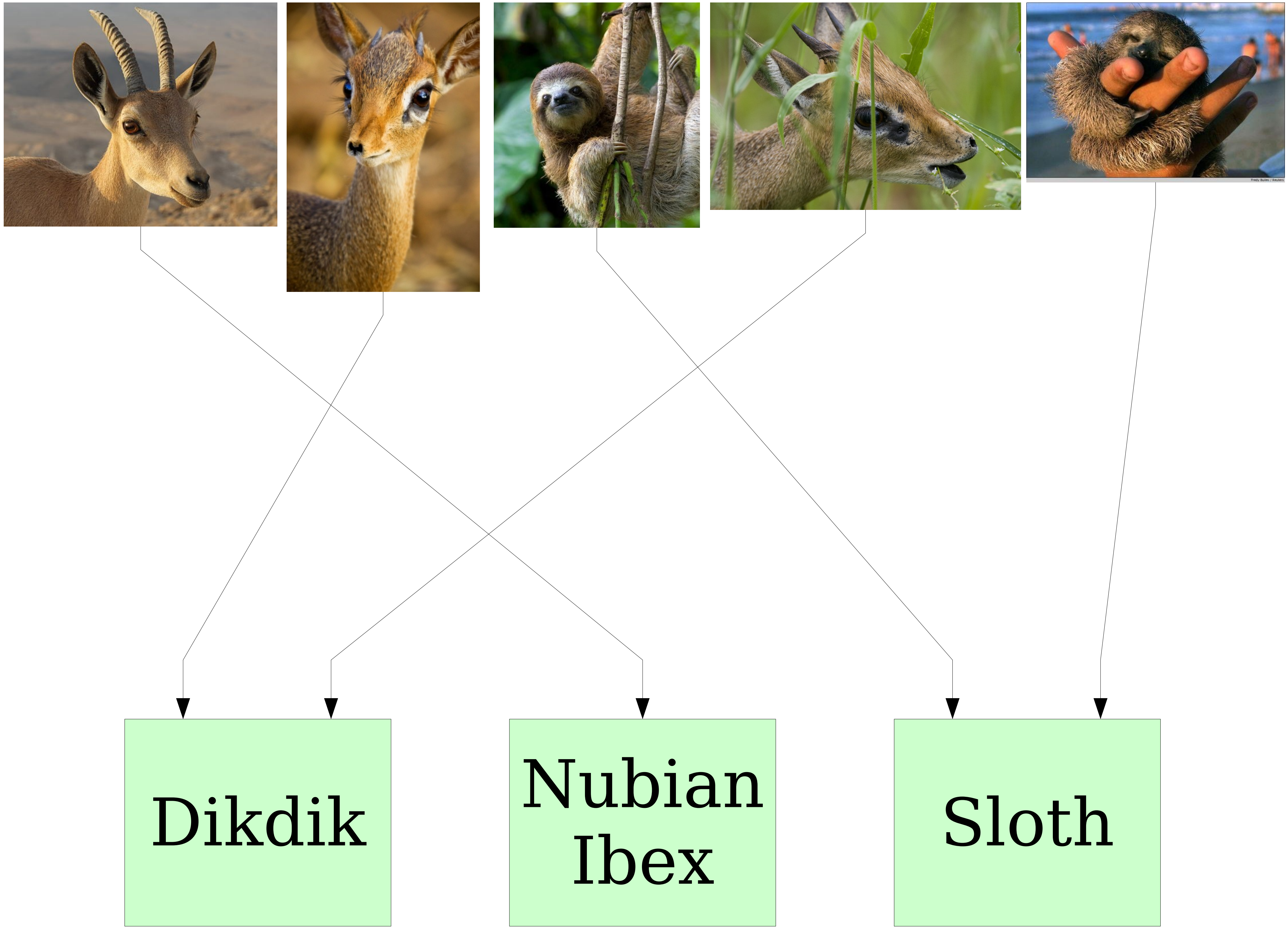
$$f(x) = 4x + 15$$

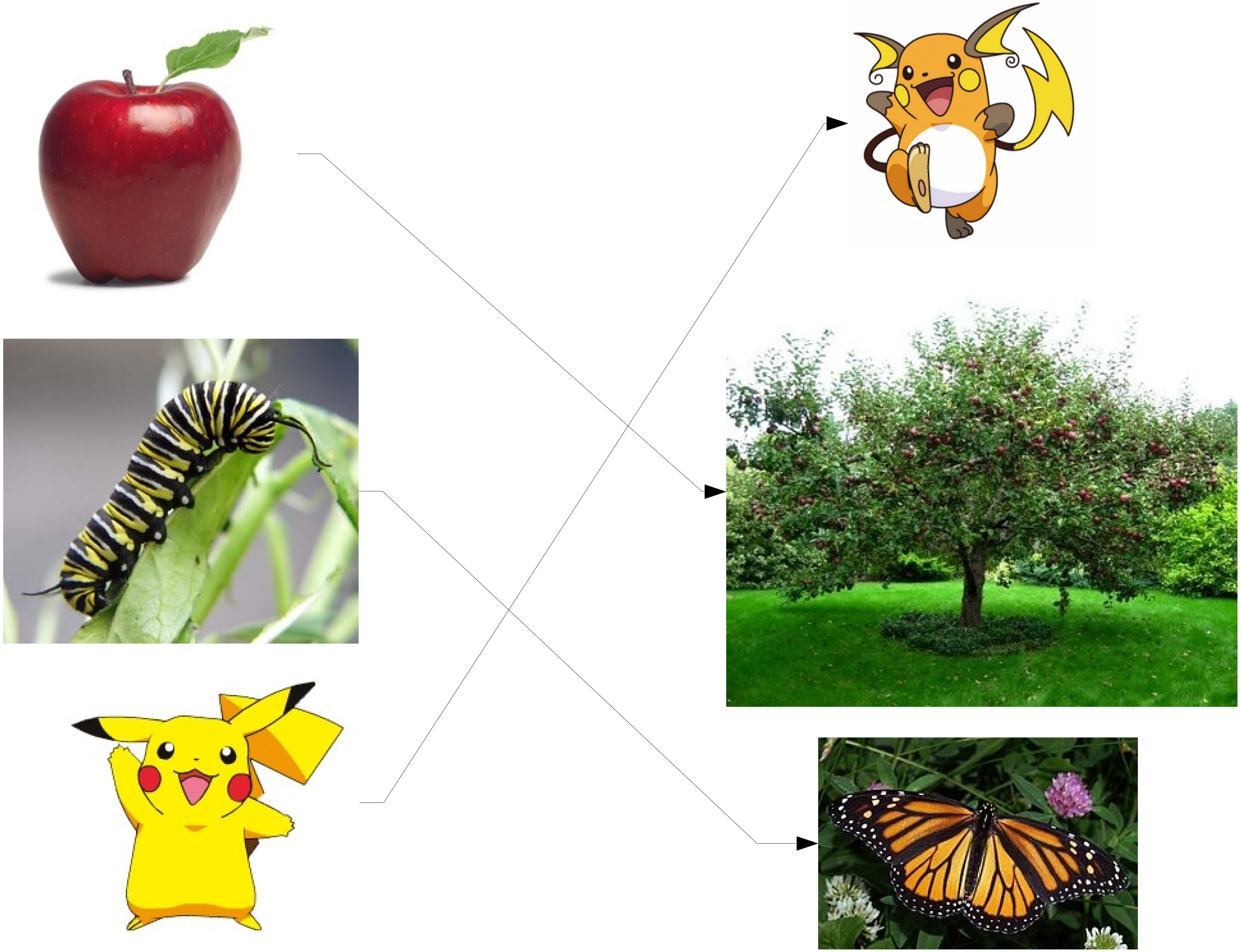
- In CS, functions are usually given by code:

```
int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;

}
```

- What sorts of functions are we going to allow from a mathematical perspective?


Fredy Builes / Reuters
Dikdik
Nubian Ibex
Sloth



… but also …

$$f(x) = x^2 + 3x - 15$$

In mathematics, functions are ***deterministic***.

That is, given the same input, a function must always produce the same output.

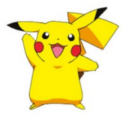
The following is a perfectly valid piece of C++ code, but it's not a valid function under our definition:

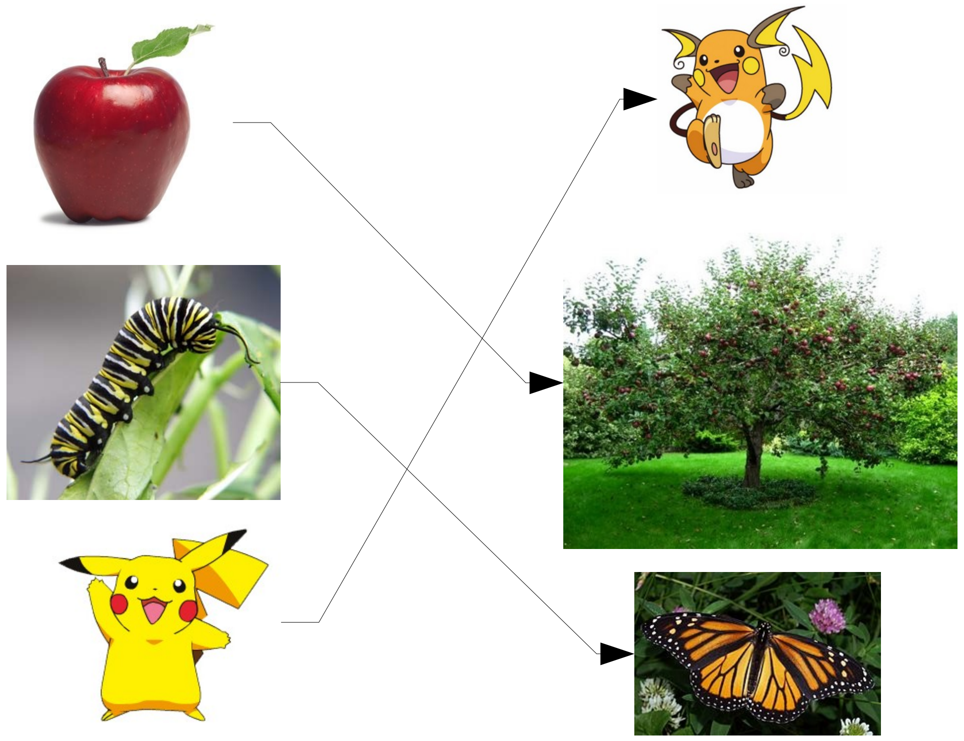
```
int randomNumber(int numOutcomes) {
    return rand() % numOutcomes;
}
```

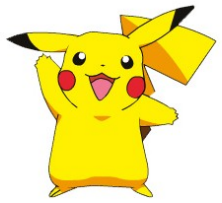
One Challenge

$$f(x) = x^2 + 2x + 5$$

$$f(\ 3\ ) = 3^2 + 3 \cdot 2 + 5 = 20$$
$$f(\ 0\ ) = 0^2 + 0 \cdot 2 + 5 = 5$$

$f($  $) = \ldots\ ?$



$f($  $) =$

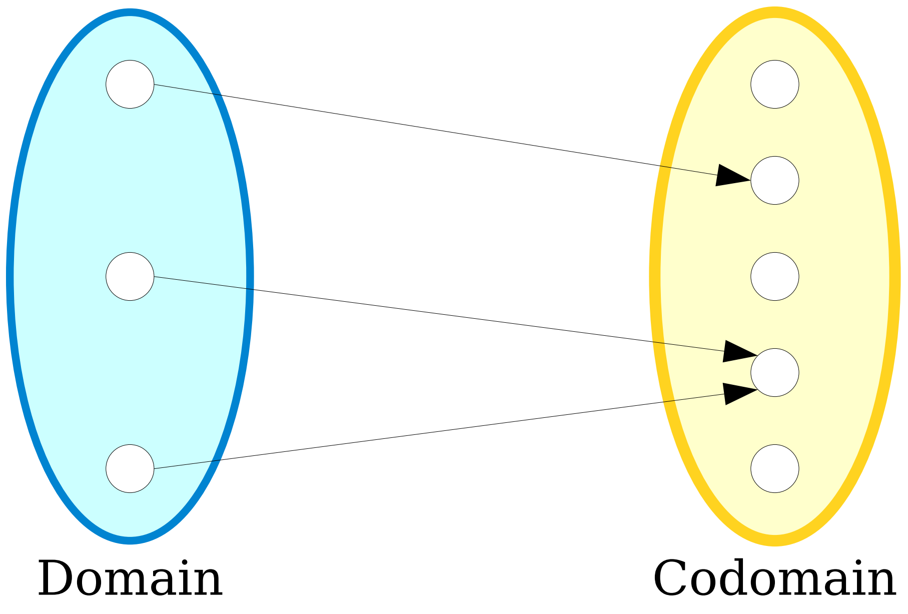
$f(137) = \ldots?$

We need to make sure we can't apply functions to meaningless inputs.

Domains and Codomains

- Every function *f* has two sets associated with it: its ***domain*** and its ***codomain***.
- A function *f* can only be applied to elements of its domain. For any *x* in the domain, *f*(*x*) belongs to the codomain.

The function must be defined for every element of the domain.



The output of the function must always be in the codomain, but not all elements of the codomain must be produced as outputs.

Domains and Codomains

- Every function *f* has two sets associated with it: its ***domain*** and its ***codomain***.
- A function *f* can only be applied to elements of its domain. For any *x* in the domain, *f*(*x*) belongs to the codomain.

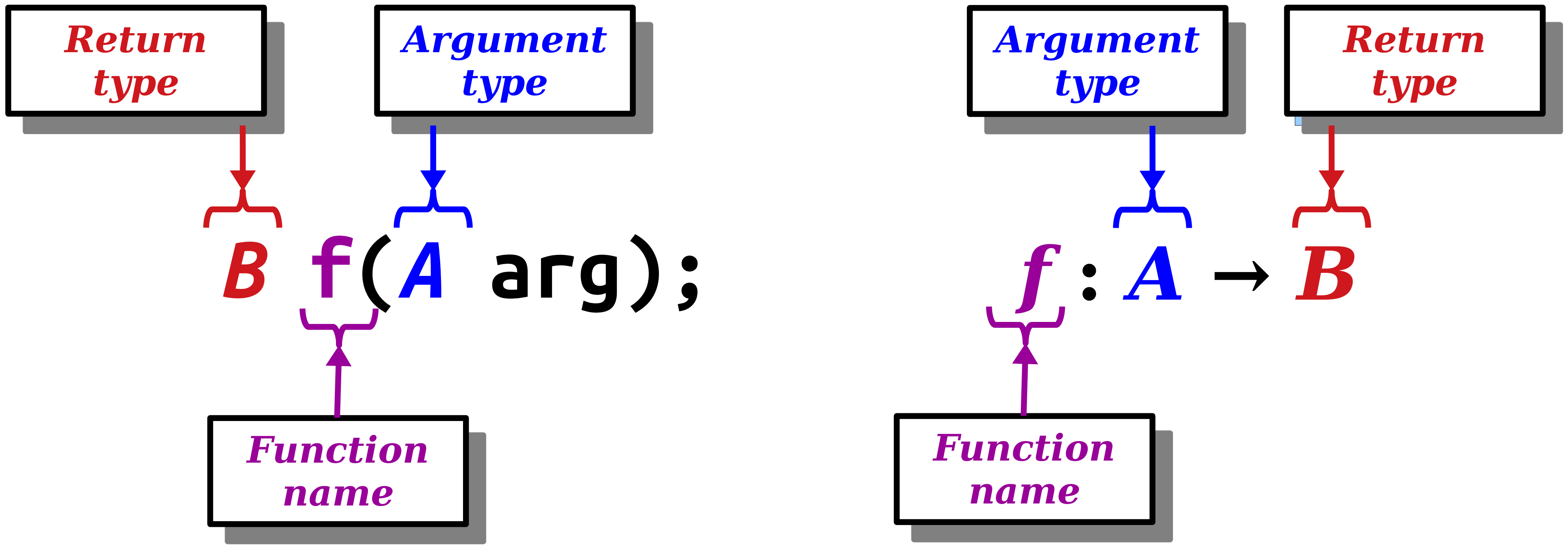
The ***domain*** of this function is ℝ. Any real number can be provided as input.

The ***codomain*** of this function is ℝ. Everything produced is a real number, but not all real numbers can be produced.

```
double absoluteValueOf(double x) {
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
}
```

Domains and Codomains

- If f is a function whose domain is A and whose codomain is B, we write $f : A \to B$.
- Think of this like a "function prototype" in C++.



Domains and Codomains

- If f is a function whose domain is A and whose codomain is B, we write $\boldsymbol{f : A \to B}$.
- Think of this like a "function prototype" in C++.

$$\overbrace{B}^{\textbf{Codomain}}\ \underbrace{\texttt{f}}_{\textbf{Function name}}\texttt{(}\overbrace{A}^{\textbf{Domain}}\ \texttt{arg);}$$

$$\underbrace{f}_{\textbf{Function name}} : \overbrace{A}^{\textbf{Domain}} \to \overbrace{B}^{\textbf{Codomain}}$$

Domains and Codomains

- Usually, when working with functions, you pick the domain and codomain *before* defining the rule for the function.
- Think programming: you usually know what types of things you're working with before you know how they work.

The Official Rules for Functions

- Formally speaking, we say that $f : A \to B$ if the following two rules hold.
- First, f must be obey its domain/codomain rules:

$$\forall a \in A.\ \exists b \in B.\ f(a) = b$$

(*"Every input in A maps to some output in B."*)

- Second, f must be deterministic:

$$\forall a_1 \in A.\ \forall a_2 \in A.\ (a_1 = a_2 \to f(a_1) = f(a_2))$$

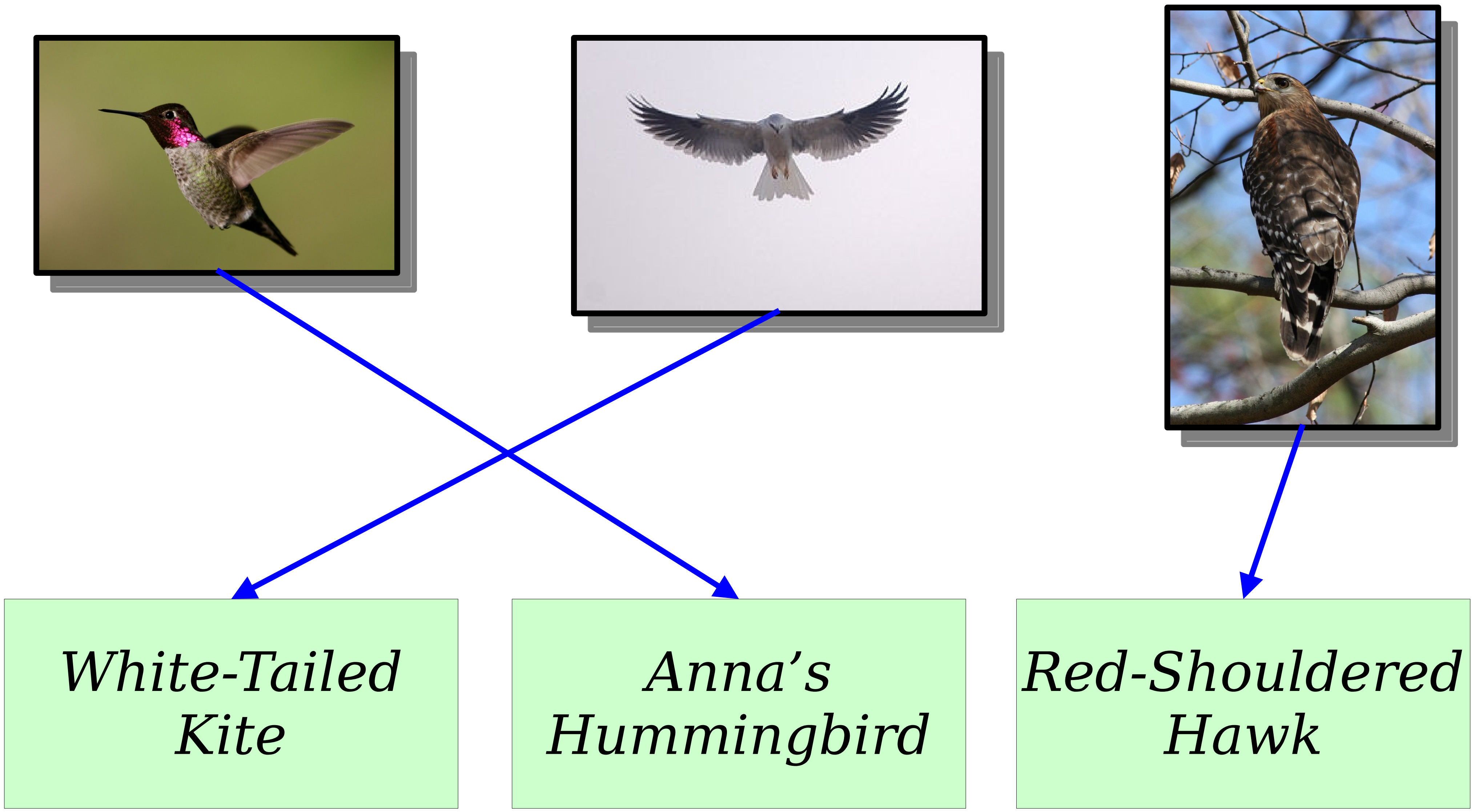
(*"Equal inputs produce equal outputs."*)

- If you're ever curious about whether something is a function, look back at these rules and check! For example:
  - Can a function have an empty domain?
  - Can a function have an empty codomain?

Defining Functions

Defining Functions

- To define a function, you need to
  - specify the domain,
  - specify the codomain, and
  - give a ***rule*** used to evaluate the function.
- All three pieces are necessary.
  - We need to domain to know what the function can be applied to.
  - We need to codomain to know what the output space is.
  - We need the rule to be able to evaluate the function.
- There are many ways to do this. Let’s go over a few examples.



Functions can be defined as a ***picture***.
Draw the domain and codomain explicitly.
Then, add arrows to show the outputs.

$$f : \mathbb{Z} \to \mathbb{Z}, \text{ where}$$

$$f(x) = x^2 + 3x - 15$$

Functions can be defined as a ***rule***.
Be sure to explicitly state what the
domain and codomain are!

$f : \mathbb{Z} \to \mathbb{N}$, where

$$f(n) = \begin{cases} n & \text{if } n \geq 0 \\ -n & \text{if } n \leq 0 \end{cases}$$

Some rules are given ***piecewise***. We select which rule to apply based on the conditions on the right. (Just make sure at least one condition applies and that all applicable conditions give the same result!)

Some Nuances

$$f(x) = \frac{x+2}{x+1}$$

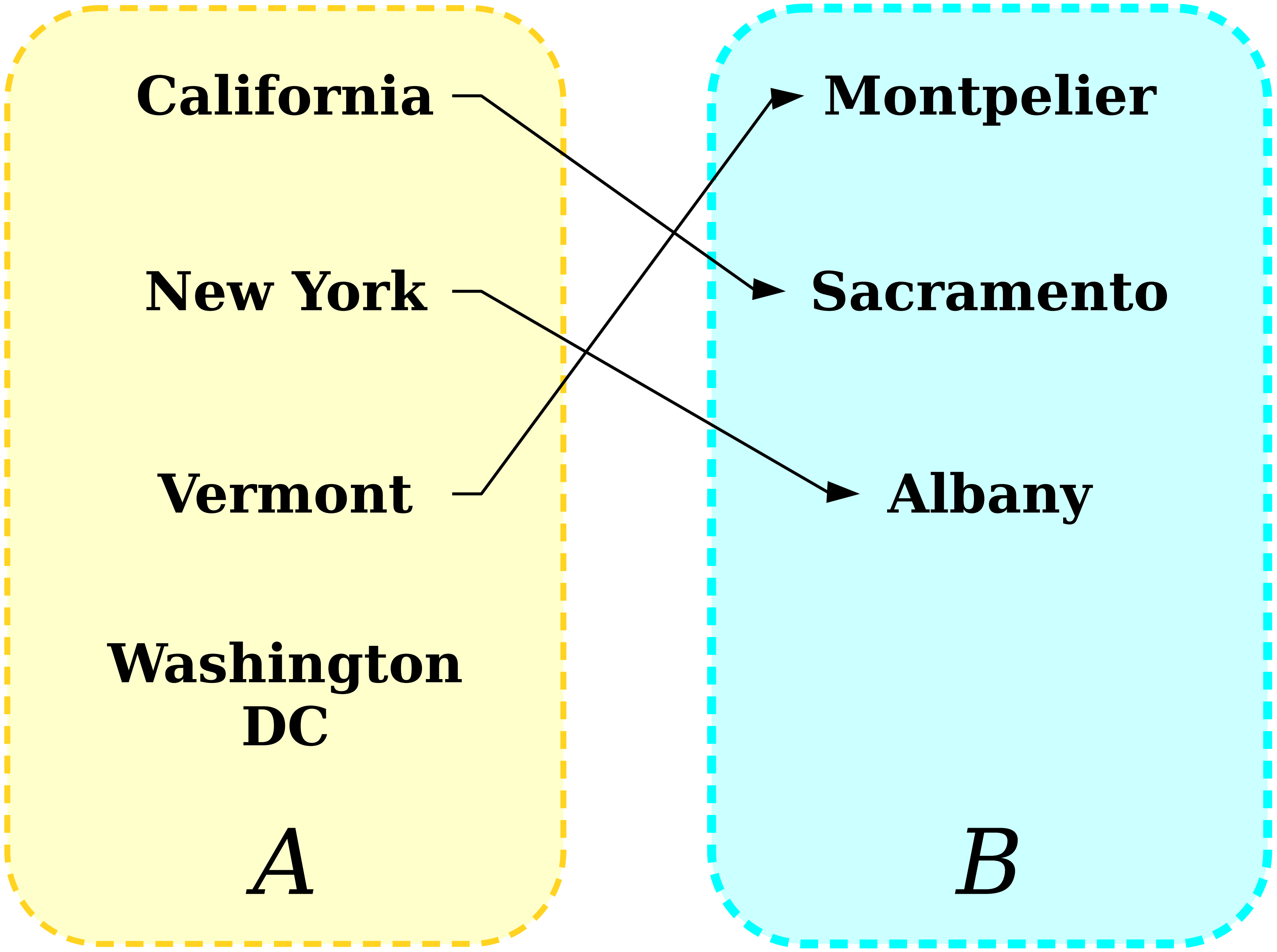
This expression isn't defined when x = -1, so f isn't defined over its full domain. We therefore don't consider it to be a function.

Is this a function from $\mathbb{R}$ to $\mathbb{R}$?

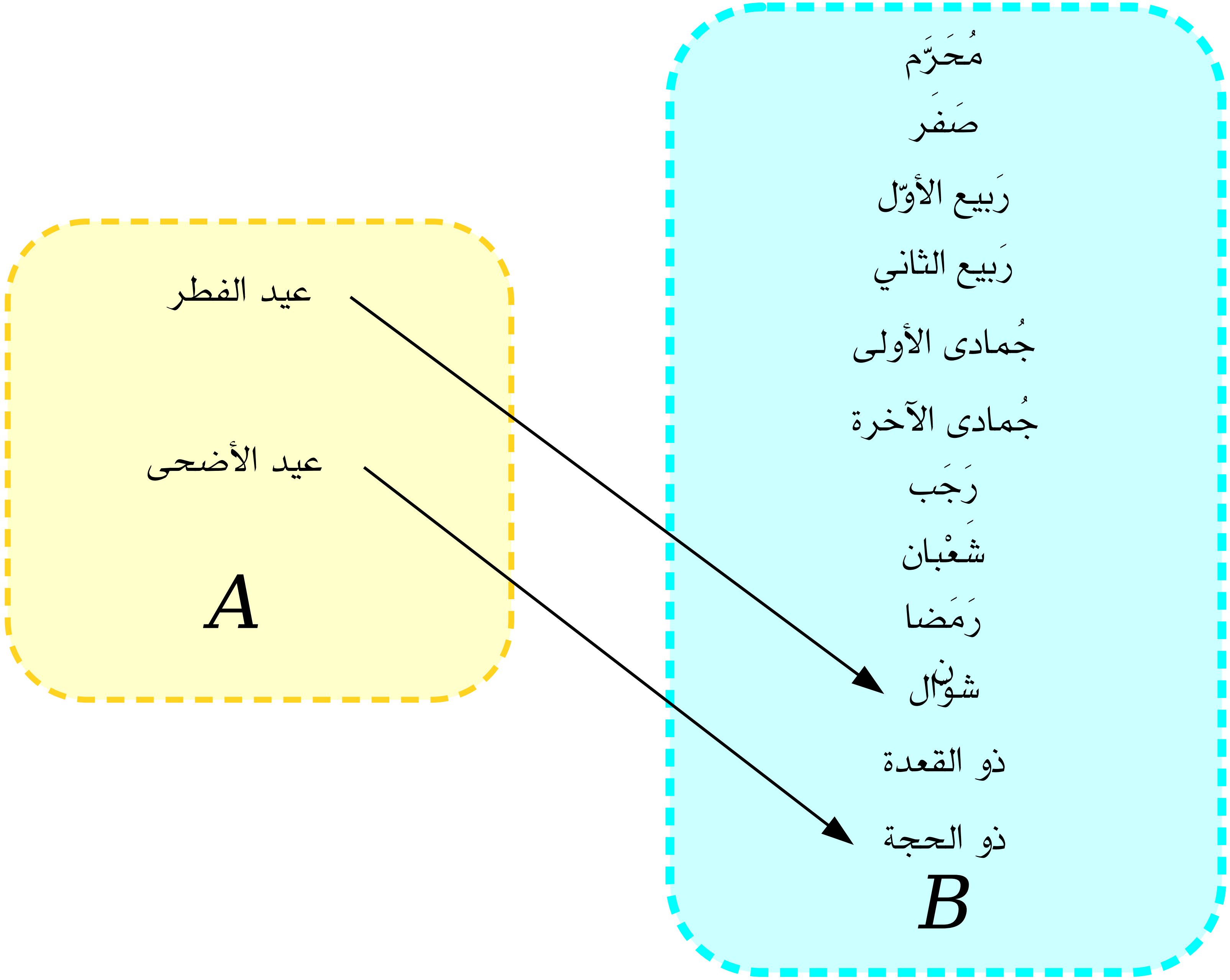
$$f(x) = \frac{x+2}{x+1}$$

Yep, it's a function! Every natural number maps to some real number.

Is this a function from $\mathbb{N}$ to $\mathbb{R}$?



Is this a function from A to B?



Is this a function from A to B?

```
int pizkwat(int input) {
    int steps = 0;
    while (input != 0) {
        input -= 2;
        steps++;
    }
    return steps;
}
```

This code never produces a value when called on the input 137. It's therefore not defined for all elements of the domain, so it's not a function in the mathematical sense.

Is this a function from $\mathbb{Z}$ to $\mathbb{Z}$?

Time-Out for Announcements!

Gradescope Tagging

- When you upload a PDF to Gradescope, please make sure to tag the pages that have your problem answers on them.
- The ***altruistic*** reason: if you don’t do this, the TAs have to do it for you, and across 155 submissions that adds up to hours of extra work.
- The ***selfish*** reason: if you don’t tag the page containing a problem, Gradescope marks it as though you didn’t submit it, and the TAs might give you no points because they thought you didn’t submit anything.
- You can tag pages after you submit, so if you submit and then realize you forgot to tag things you can always go back and fix it.

Problem Set One Solutions

- We’ve just posted solutions to Problem Set One. They’re linked from the main PS1 page.
- We recommend you read over our solution set before finishing PS2.
  - You’ll get to see examples of polished written proofs.
  - Each problem has a “Why We Asked This Question” section, which gives some context.
  - We may have solved the problem differently than you, and this will give you more perspectives to use.
- We’ll aim to have PS1 graded and returned by tomorrow afternoon.

Your Questions

“What skills/courses do you recommend to have the ‘chops’ needed to get a job in tech after we graduate?”

Most software positions are aimed at generalist programmers. (Some job postings ask for specific qualifications – ignore them and apply anyway.) Write a lot of code across your classes. You don't necessarily need to do side projects just to prove a point, though if there's something you'd like to try your hand at building that you yourself are personally excited about, go for it!

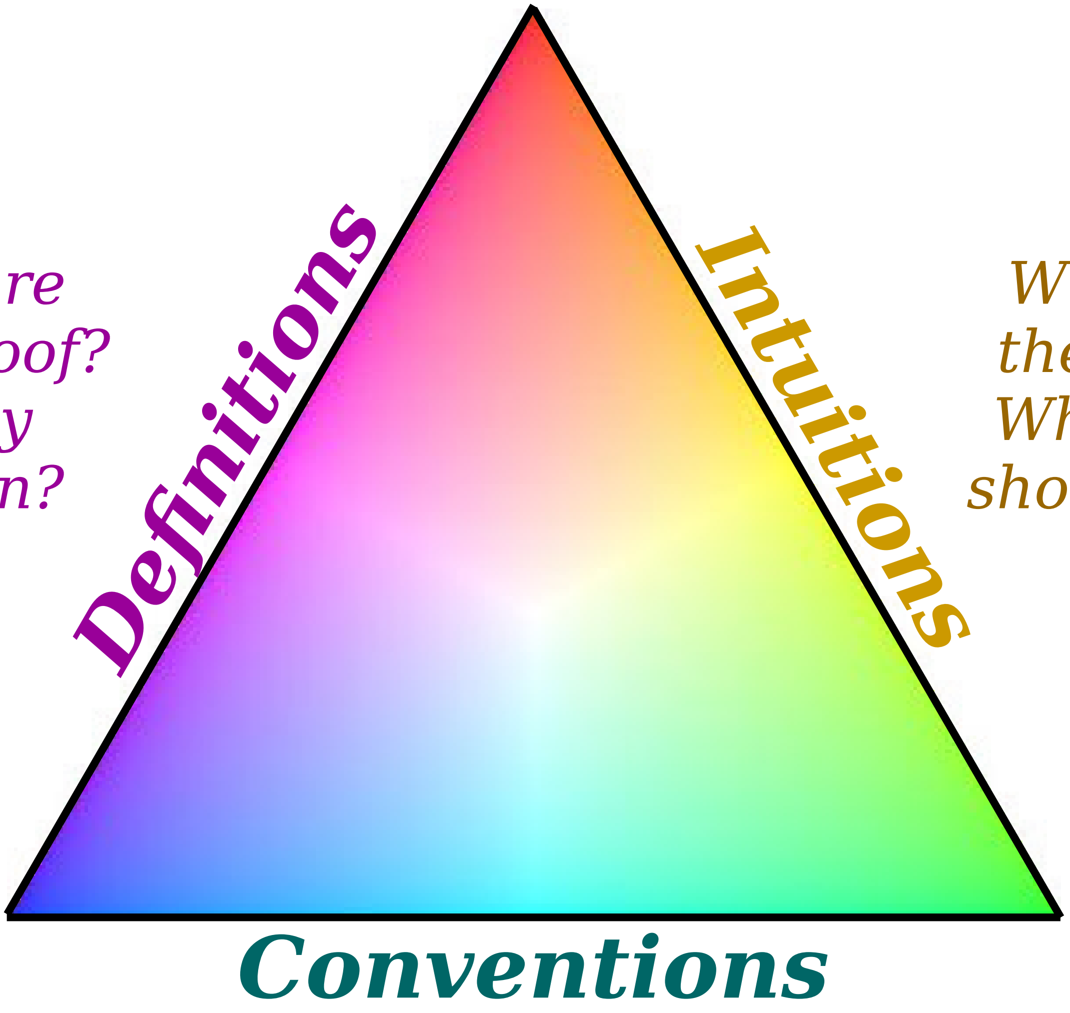
Also, make sure you understand the theory of whatever you concentrate in. The field will change rapidly in a short time period. Knowing the latest/greatest will help initially, but you will need to adapt and deep knowledge pays huge dividends.

Pick some language or some technology that you feel very comfortable expressing yourself in. It will make interviewing easier and will help you get into the mindset of how to learn technologies as systems.

Back to CS103!

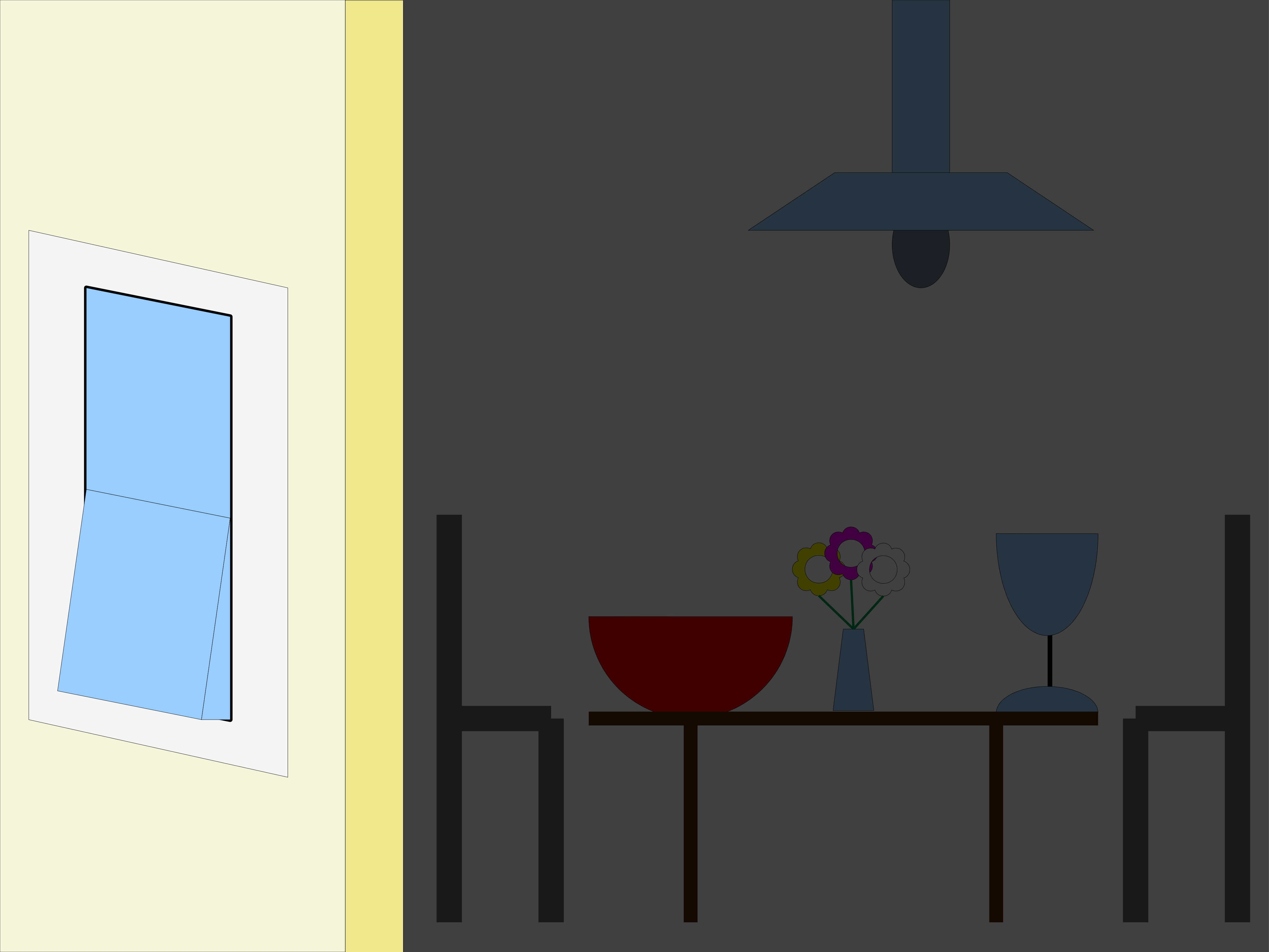
Special Types of Functions

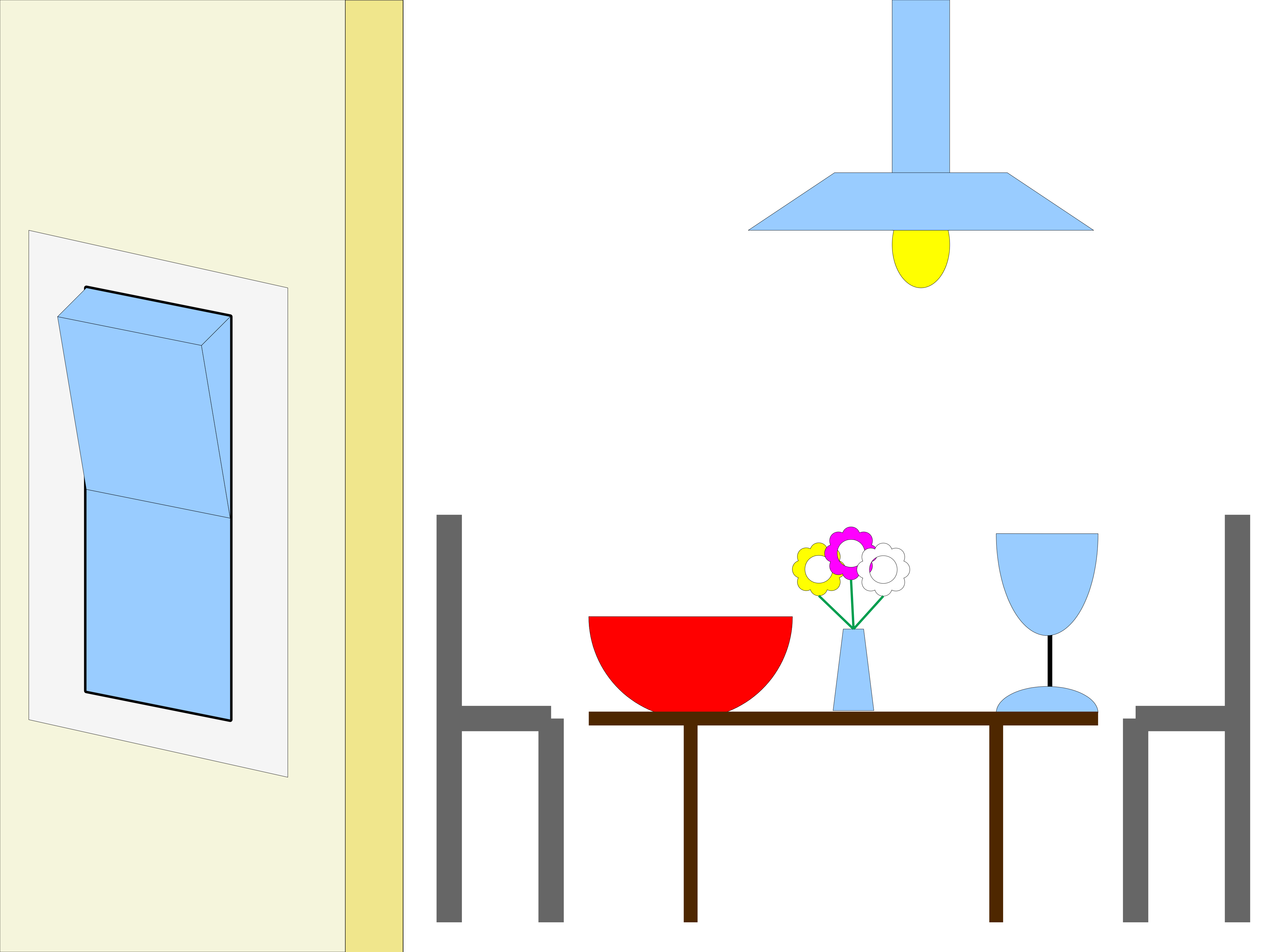
What terms are used in this proof? What do they formally mean?



What does this theorem mean? Why, intuitively, should it be true?

What is the standard format for writing a proof? What are the techniques for doing so?





Undoing by Doing Again

- Some operations invert themselves. For example:
  - Flipping a switch twice is the same as not flipping it at all.
  - In first-order logic, $\neg\neg A$ is equivalent to A.
  - In algebra, $-(-x) = x$.
  - In set theory, $(A \Delta B) \Delta B = A$. *(Yes, really!)*
- Operations with these properties are surprisingly useful in CS theory and come up in a bunch of contexts.
  - Storing compressed approximations of sets (XOR filters).
  - Building encryption systems (symmetric block ciphers).
  - Transmitting a large file to multiple receivers (fountain codes).

Involutions

- A function $f : A \to A$ from a set back to itself is called an ***involution*** if the following first-order logic statement is true about f:

$$\forall x \in A.\ f(f(x)) = x.$$

("Applying f twice is equivalent to not applying f at all.")

- Involutions have lots of interesting properties. Let's explore them and see what we can find.

Involutions

- Which of the following are involutions?
  - $f : \mathbb{Z} \to \mathbb{Z}$ defined as $f(x) = x$.
  - $f : \mathbb{Z} \to \mathbb{Z}$ defined as $f(x) = -x$.
  - $f : \mathbb{R} \to \mathbb{R}$ defined as $f(x) = {}^1\!/_x$.
  - $f : \mathbb{N} \to \mathbb{N}$ defined as follows:

$$f(n) = \begin{cases} n+1 & \text{if } n \text{ is even} \\ n-1 & \text{if } n \text{ is odd} \end{cases}$$

A function $f : A \to A$ is called an ***involution*** if the following first-order logic statement is true about f:

$$\forall x \in A.\ f(f(x)) = x.$$

Involutions

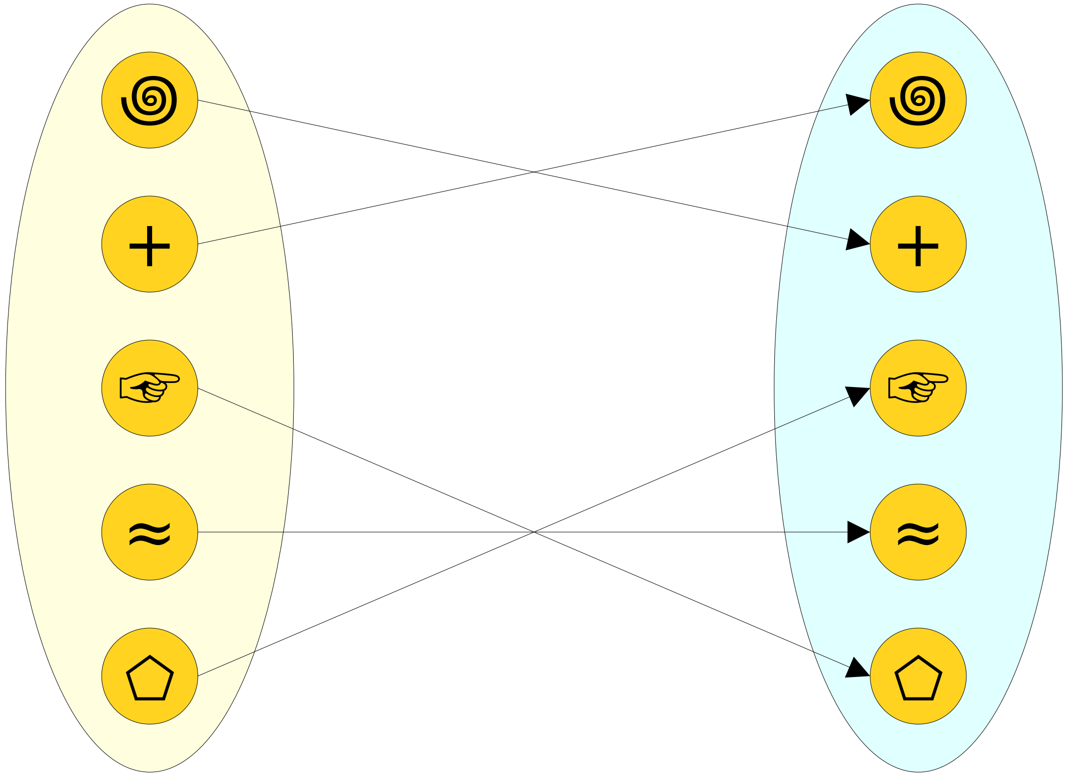
- Which of the following are involutions?
  - $f : \mathbb{Z} \to \mathbb{Z}$ defined as $f(x) = x$. *Yep!*
  - $f : \mathbb{Z} \to \mathbb{Z}$ defined as $f(x) = -x$. *Yep!*
  - $f : \mathbb{R} \to \mathbb{R}$ defined as $f(x) = {}^1/_x$. *Not a function!*
  - $f : \mathbb{N} \to \mathbb{N}$ defined as follows: *Yep!*

$$f(n) = \begin{cases} n+1 & \text{if } n \text{ is even} \\ n-1 & \text{if } n \text{ is odd} \end{cases}$$

A function $f : A \to A$ is called an ***involution*** if the following first-order logic statement is true about f:

$$\boldsymbol{\forall x \in A.\ f(f(x)) = x.}$$

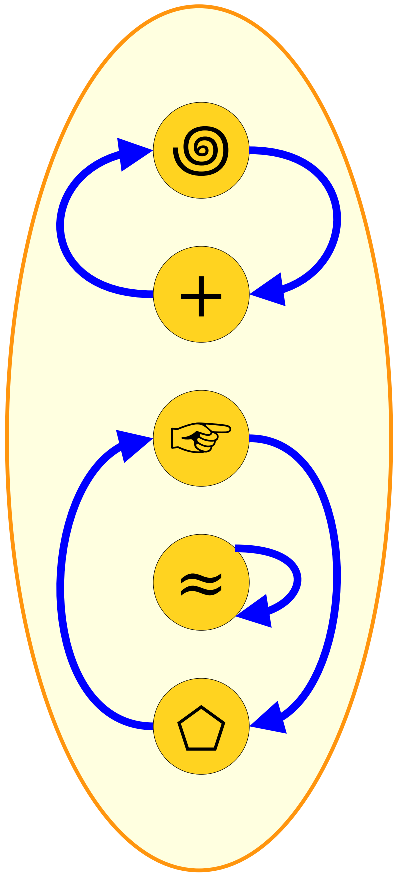
Involutions, Visually



A function $f : A \to A$ is called an ***involution*** if the following first-order logic statement is true about f:

$$\forall x \in A.\ f(f(x)) = x.$$

Involutions, Visually



A function $f : A \to A$ is called an ***involution*** if the following first-order logic statement is true about f:

$$\forall x \in A.\ f(f(x)) = x.$$

Proofs on Involutions

Theorem: The function $f : \mathbb{Z} \to \mathbb{Z}$ defined as

$$f(n) = \begin{cases} n+1 & \text{if } n \text{ is even} \\ n-1 & \text{if } n \text{ is odd} \end{cases}$$

is an involution.

Proof:

What does it mean for f to be an involution?

$$\forall n \in \mathbb{Z}.\ f(f(n)) = n.$$

Therefore, we'll have the reader pick some $n \in \mathbb{Z}$, then argue that $f(f(n)) = n$.

Theorem: The function $f : \mathbb{Z} \to \mathbb{Z}$ defined as

$$f(n) = \begin{cases} n+1 & \text{if } n \text{ is even} \\ n-1 & \text{if } n \text{ is odd} \end{cases}$$

is an involution.

Proof: Pick some $n \in \mathbb{Z}$. We need to show that $f(f(n)) = n$. To do so, we consider two cases.

Case 1: n is even. Then $f(n) = n+1$, which is odd. This means that $f(f(n)) = f(n+1) = (n+1) - 1 = n$.

Case 2: n is odd. Then $f(n) = n - 1$, which is even. Then we see that $f(f(n)) = f(n - 1) = (n - 1) + 1 = n$.

In either case, we see that $f(f(n)) = n$, which is what we need to show. ■

This proof contains no first-order logic syntax (quantifiers, connectives, etc.). It's written in plain English, just as usual.

	To ***prove*** that this is true…	
$\forall x.\ A$	Have the reader pick an arbitrary x. We then prove A is true for that choice of x.	

Theorem: The function $f : \mathbb{N} \to \mathbb{N}$ defined as $f(n) = n^2$ is not an involution.

What does it mean for f to be an involution?

$$\forall n \in \mathbb{N}.\ f(f(n)) = n.$$

What is the negation of this statement?

$$\neg\forall n \in \mathbb{N}.\ f(f(n)) = n$$
$$\exists n \in \mathbb{N}.\ \neg(f(f(n)) = n)$$
$$\exists n \in \mathbb{N}.\ f(f(n)) \neq n$$

Therefore, we need to pick some concrete choice of n such that $f(f(n)) \neq n$.

Theorem: The function $f : \mathbb{N} \to \mathbb{N}$ defined as $f(n) = n^2$ is not an involution.

Proof: We need to show that there is some $n \in \mathbb{N}$ where $f(f(n)) \neq n$.

Pick $n = 2$. Then

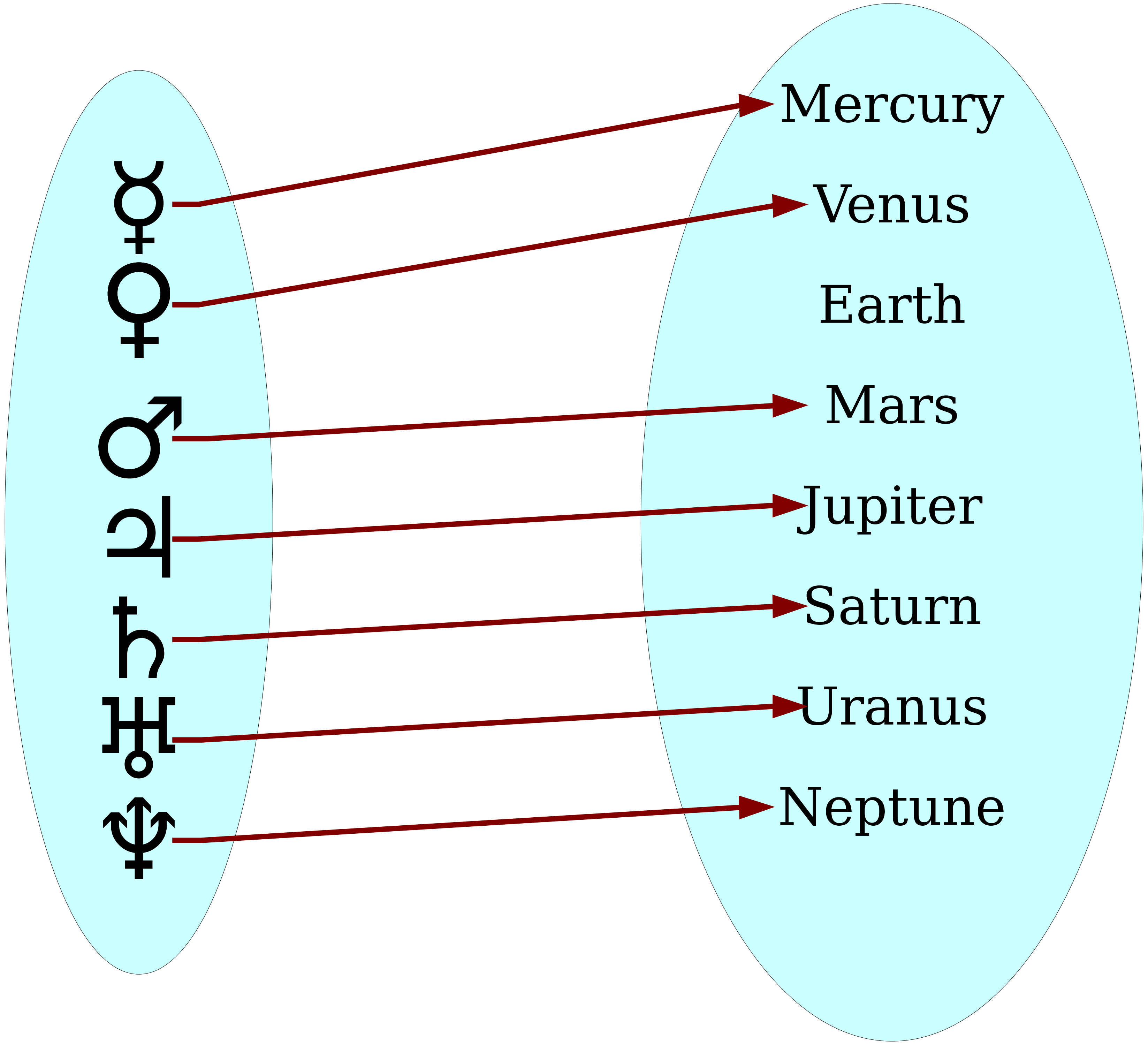
$$\begin{aligned} f\,(f(n)) &= f(f(2)) \\ &= f(4) \\ &= 16, \end{aligned}$$

which means that $f(f(n)) \neq 2$, as required. ■

This proof contains no first-order logic syntax (quantifiers, connectives, etc.). It's written in plain English, just as usual.

	To ***prove*** that this is true…	
$\forall x.\ A$	Have the reader pick an arbitrary x. We then prove A is true for that choice of x.	
$\exists x.\ A$	Find an x where A is true. Then prove that A is true for that specific choice of x.	
$\neg A$	Simplify the negation, then consult this table on the result.	

Another Class of Functions


☿
♀
♂
♃
♄
♅
♆
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

Injective Functions

- A function $f : A \to B$ is called ***injective*** (or ***one-to-one***) if the following statement is true about f:

$$\forall a_1 \in A.\ \forall a_2 \in A.\ (a_1 \neq a_2 \to f(a_1) \neq f(a_2))$$

(*"If the inputs are different, the outputs are different."*)

- The following first-order definition is equivalent *(why?)* and is often useful in proofs.

$$\forall a_1 \in A.\ \forall a_2 \in A.\ (f(a_1) = f(a_2) \to a_1 = a_2)$$

(*"If the outputs are the same, the inputs are the same."*)

- A function with this property is called an ***injection***.
- How does this compare to our second rule for functions?

Injections

- Let be the set of all CS103 students. Which of the following are injective?
  - f : $\to \mathbb{N}$ where $f(x)$ is x's Stanford ID number.
  - f : $\to$, where is the set of all countries and $f(x)$ is x's country of birth.
  - f : $\to$, where is the set of all given (first) names, where $f(x)$ is x's given (first) name.

$f : A \to B$ is ***injective*** when either equivalent statement is true:

$$\forall x_1 \in A.\ \forall x_2 \in A.\ (x_1 \neq x_2 \to f(x_1) \neq f(x_2))$$
$$\forall x_1 \in A.\ \forall x_2 \in A.\ (f(x_1) = f(x_2) \to x_1 = x_2)$$

Next Time

- ***First-Order Assumptions***
  - The difference between assuming something is true and proving something is true.
- ***Connecting Function Types***
  - Involutions, injections, and surjections are related to one another. How?
- ***Function Composition***
  - Sequencing functions together.