

## Well defined functions

Recall that a function is defined by its (1) domain, (2) codomain, and (3) rule assigning each element in the domain exactly one element in the codomain. The domain and codomain are nonempty sets. The rule can be depicted as a table, formula, English description, etc.

A function can *fail to be well-defined* if there is some domain element where the function rule doesn't give a unique codomain element. For example, the function rule might lead to more than one potential image, or to an image outside of the codomain.

*Example:*  $f_A : \mathbb{R}^+ \rightarrow \mathbb{Q}$  with  $f_A(x) = x$  is **not** a well-defined function because

*Example:*  $f_B : \mathbb{Q} \rightarrow \mathbb{Z}$  with  $f_B\left(\frac{p}{q}\right) = p + q$  is **not** a well-defined function because

*Example:*  $f_C : \mathbb{Z} \rightarrow \mathbb{R}$  with  $f_C(x) = \frac{x}{|x|}$  is **not** a well-defined function because

## Injective function definition

**Definition :** A function  $f : D \rightarrow C$  is **one-to-one** (or injective) means for every  $a, b$  in the domain  $D$ , if  $f(a) = f(b)$  then  $a = b$ .

Formally,  $f : D \rightarrow C$  is one-to-one means \_\_\_\_\_.

## Injective functions visually

Informally, a function being one-to-one means “no duplicate images”.

## Surjective function definition

**Definition:** A function  $f : D \rightarrow C$  is **onto** (or surjective) means for every  $b$  in the codomain, there is an element  $a$  in the domain with  $f(a) = b$ .

Formally,  $f : D \rightarrow C$  is onto means \_\_\_\_\_.

## Surjective functions visually

Informally, a function being onto means “every potential image is an actual image”.

## Bijection definition

**Definition :** A function  $f : D \rightarrow C$  is a **bijection** means that it is both one-to-one and onto. The **inverse** of a bijection  $f : D \rightarrow C$  is the function  $g : C \rightarrow D$  such that  $g(b) = a$  iff  $f(a) = b$ .

# Predicate definition

**Definition:** A **predicate** is a function from a given set (domain) to  $\{T, F\}$ .

A predicate can be applied, or **evaluated** at, an element of the domain.

Usually, a predicate *describes a property* that domain elements may or may not have.

Two predicates over the same domain are **equivalent** means they evaluate to the same truth values for all possible assignments of domain elements to the input. In other words, they are equivalent means that they are equal as functions.

To define a predicate, we must specify its domain and its value at each domain element. The rule assigning truth values to domain elements can be specified using a formula, English description, in a table (if the domain is finite), or recursively (if the domain is recursively defined).

## Predicate truth set definition

**Definition:** The **truth set** of a predicate is the collection of all elements in its domain where the predicate evaluates to  $T$ .

Notice that specifying the domain and the truth set is sufficient for defining a predicate.

## Predicate truth set example

The truth set for the predicate  $V(x)$  is \_\_\_\_\_.

The truth set for the predicate  $N(x)$  is \_\_\_\_\_.

The truth set for the predicate  $Mystery(x)$  is \_\_\_\_\_.

# Defining functions more examples

Let's practice with functions related to some of our applications so far.

Recall: We model the collection of user ratings of the four movies Dune, Oppenheimer, Barbie, Nimona as the set  $\{-1, 0, 1\}^4$ . One function that compares pairs of ratings is

$$d_0 : \{-1, 0, 1\}^4 \times \{-1, 0, 1\}^4 \rightarrow \mathbb{R}$$

given by

$$d_0((x_1, x_2, x_3, x_4), (y_1, y_2, y_3, y_4)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + (x_4 - y_4)^2}$$

Notice: any ordered pair of ratings is an okay input to  $d_0$ .

Notice: there are (at most)

$$(3 \cdot 3 \cdot 3 \cdot 3) \cdot (3 \cdot 3 \cdot 3 \cdot 3) = 3^8 = 6561$$

many pairs of ratings. There are therefore lots and lots of real numbers that are not the output of  $d_0$ .

Recall: RNA is made up of strands of four different bases that encode genomic information in specific ways. The bases are elements of the set  $B = \{\mathbf{A}, \mathbf{C}, \mathbf{U}, \mathbf{G}\}$ . The set of RNA strands  $S$  is defined (recursively) by:

$$\begin{array}{ll} \text{Basis Step:} & \mathbf{A} \in S, \mathbf{C} \in S, \mathbf{U} \in S, \mathbf{G} \in S \\ \text{Recursive Step:} & \text{If } s \in S \text{ and } b \in B, \text{ then } sb \in S \end{array}$$

where  $sb$  is string concatenation.

**Pro-tip:** informal definitions sometime use  $\dots$  to indicate “continue the pattern”. Often, to make this pattern precise we use recursive definitions.

Name	Domain	Codomain	Rule	Example
<i>rnalen</i>	$S$	$\mathbb{Z}^+$	<div> <div>Basis Step:</div> <div>If <math>b \in B</math> then <math>rnalen(b) = 1</math></div> <div>Recursive Step:</div> <div>If <math>s \in S</math> and <math>b \in B</math>, then</div> <div><math>rnalen(sb) = 1 + rnalen(s)</math></div> </div>	<div> <div><math>rnalen(\mathbf{AC}) \stackrel{\text{rec step}}{=} 1 + rnalen(\mathbf{A})</math></div> <div><math>\stackrel{\text{basis step}}{=} 1 + 1 = 2</math></div> </div>
<i>basecount</i>	$S \times B$	$\mathbb{N}$	<div> <div>Basis Step:</div> <div>If <math>b_1 \in B, b_2 \in B</math> then</div> <div><math>basecount( (b_1, b_2) ) =</math></div> <div> <math display="block">\begin{cases} 1 &amp; \text{when } b_1 = b_2 \\ 0 &amp; \text{when } b_1 \neq b_2 \end{cases}</math> </div> <div>Recursive Step:</div> <div>If <math>s \in S, b_1 \in B, b_2 \in B</math></div> <div><math>basecount( (sb_1, b_2) ) =</math></div> <div> <math display="block">\begin{cases} 1 + basecount( (s, b_2) ) &amp; \text{when } b_1 = b_2 \\ basecount( (s, b_2) ) &amp; \text{when } b_1 \neq b_2 \end{cases}</math> </div> </div>	$basecount( (\mathbf{ACU}, \mathbf{C}) ) =$
“2 to the power of”	$\mathbb{N}$	$\mathbb{N}$	<div> <div>Basis Step:</div> <div><math>2^0 = 1</math></div> <div>Recursive Step:</div> <div>If <math>n \in \mathbb{N}, 2^{n+1} =</math></div> </div>	
“ $b$ to the power of $i$ ”	$\mathbb{Z}^+ \times \mathbb{N}$	$\mathbb{N}$	<div> <div>Basis Step:</div> <div><math>b^0 = 1</math></div> <div>Recursive Step:</div> <div>If <math>i \in \mathbb{N}, b^{i+1} = b \cdot b^i</math></div> </div>	

$2^0 = 1$ 
 $2^1 = 2$ 
 $2^2 = 4$ 
 $2^3 = 8$ 
 $2^4 = 16$ 
 $2^5 = 32$ 
 $2^6 = 64$ 
 $2^7 = 128$ 
 $2^8 = 256$ 
 $2^9 = 512$ 
 $2^{10} = 1024$

# Definitions set prereqs

Term	Notation	Example(s)	We say in English ...
all reals	$\mathbb{R}$		The (set of all) real numbers (numbers on the number line)
all integers	$\mathbb{Z}$		The (set of all) integers (whole numbers including negatives, zero, and positives)
all positive integers	$\mathbb{Z}^+$		The (set of all) strictly positive integers
all natural numbers	$\mathbb{N}$		The (set of all) natural numbers. <b>Note:</b> we use the convention that 0 is a natural number.

# Definitions functions prereqs

Term	Notation	Example(s)	We say in English ...
sequence	$x_1, \dots, x_n$		A sequence $x_1$ to $x_n$
summation	$\sum_{i=1}^n x_i$ or $\sum_{i=1}^n x_i$		The sum of the terms of the sequence $x_1$ to $x_n$
piecewise definition	rule	$f(x) = \begin{cases} \text{rule 1 for } x & \text{when COND 1} \\ \text{rule 2 for } x & \text{when COND 2} \end{cases}$	Define $f$ of $x$ to be the result of applying rule 1 to $x$ when condition COND 1 is true and the result of applying rule 2 to $x$ when condition COND 2 is true. This can be generalized to having more than two conditions (or cases).
function application		$f(7)$ $f(z)$ $f(g(z))$	$f$ of 7 <b>or</b> $f$ applied to 7 <b>or</b> the image of 7 under $f$ $f$ of $z$ <b>or</b> $f$ applied to $z$ <b>or</b> the image of $z$ under $f$ $f$ of $g$ of $z$ <b>or</b> $f$ applied to the result of $g$ applied to $z$
absolute value	$ -3 $		The absolute value of $-3$
square root	$\sqrt{9}$		The non-negative square root of 9

**Pro-tip:** the meaning of two vertical lines  $| \quad |$  depends on the data-types of what's between the lines. For example, when placed around a number, the two vertical lines represent absolute value. We've seen a single vertical line  $|$  used as part of set builder definitions to represent "such that". Again, this is (one of the many reasons) why is it very important to declare the data-type of variables before we use them.

# Defining functions

**New! Defining functions** A function is defined by its (1) domain, (2) codomain, and (3) rule assigning each element in the domain exactly one element in the codomain.

The domain and codomain are nonempty sets.

The rule can be depicted as a table, formula, piecewise definition, or English description.

The notation is

“Let the function  $\text{FUNCTION-NAME}: \text{DOMAIN} \rightarrow \text{CODOMAIN}$  be given by  
 $\text{FUNCTION-NAME}(x) = \dots$  for every  $x \in \text{DOMAIN}$ ”.

or

“Consider the function  $\text{FUNCTION-NAME}: \text{DOMAIN} \rightarrow \text{CODOMAIN}$  defined as  
 $\text{FUNCTION-NAME}(x) = \dots$  for every  $x \in \text{DOMAIN}$ ”.

Example: The absolute value function

**Domain**

**Codomain**

**Rule**

## Defining functions ratings

Recall our representation of Netflix users' ratings of movies as  $n$ -tuples, where  $n$  is the number of movies in the database. Each component of the  $n$ -tuple is  $-1$  (didn't like the movie),  $0$  (neutral rating or didn't watch the movie), or  $1$  (liked the movie).

Consider the ratings  $P_1 = (-1, 0, 1, 0)$ ,  $P_2 = (1, 1, -1, 0)$ ,  $P_3 = (1, 1, 1, 0)$ ,  $P_4 = (0, -1, 1, 0)$

Which of  $P_1$ ,  $P_2$ ,  $P_3$  has movie preferences most similar to  $P_4$ ?

One approach to answer this question: use **functions** to quantify difference among user preferences.

For example, consider the function  $d_0 : \{-1, 0, 1\}^4 \times \{-1, 0, 1\}^4 \rightarrow \mathbb{R}$  given by

$$d_0((x_1, x_2, x_3, x_4), (y_1, y_2, y_3, y_4)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + (x_4 - y_4)^2}$$



# Defining functions recursively

When the domain of a function is a *recursively defined set*, the rule assigning images to domain elements (outputs) can also be defined recursively.

Recall: The set of RNA strands  $S$  is defined (recursively) by:

$$\begin{array}{ll} \text{Basis Step:} & \mathbf{A} \in S, \mathbf{C} \in S, \mathbf{U} \in S, \mathbf{G} \in S \\ \text{Recursive Step:} & \text{If } s \in S \text{ and } b \in B, \text{ then } sb \in S \end{array}$$

where  $sb$  is string concatenation.

**Definition** (Of a function, recursively) A function  $rnalen$  that computes the length of RNA strands in  $S$  is defined by:

$$\begin{array}{llll} & & rnalen : S & \rightarrow \mathbb{Z}^+ \\ \text{Basis Step:} & \text{If } b \in B \text{ then} & rnalen(b) & = 1 \\ \text{Recursive Step:} & \text{If } s \in S \text{ and } b \in B, \text{ then} & rnalen(sb) & = 1 + rnalen(s) \end{array}$$

The domain of  $rnalen$  is

The codomain of  $rnalen$  is

Example function application:

$$rnalen(\mathbf{ACU}) =$$

*Example:* A function  $basecount$  that computes the number of a given base  $b$  appearing in a RNA strand  $s$  is defined recursively:

$$\begin{array}{llll} & & basecount : S \times B & \rightarrow \mathbb{N} \\ \text{Basis Step:} & \text{If } b_1 \in B, b_2 \in B & basecount( (b_1, b_2) ) & = \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases} \\ \text{Recursive Step:} & \text{If } s \in S, b_1 \in B, b_2 \in B & basecount( (sb_1, b_2) ) & = \begin{cases} 1 + basecount( (s, b_2) ) & \text{when } b_1 = b_2 \\ basecount( (s, b_2) ) & \text{when } b_1 \neq b_2 \end{cases} \end{array}$$