Booleans

So far we’ve encountered two types of data: numbers and strings. Now we add a third: boolean data. There are exactly two boolean values, true and false. These are denoted by true and false. Those are actually keywords, which means that they cannot be used in a definition:

\[(\text{define} \ true \ 33)\]

will generate an error, because we cannot use a keyword as the “name” part of a definition.

(George Boole, for whom boolean data is named, introduced boolean values in a book he called ‘The Laws of Thought”. Modest guy, hunh?)

1 Definitions, again

We already learned that to define a name in Racket, you write something like (define height 37). But what happens if you do that twice? What if you write something like

\[(\text{define} \ height \ 12)\]
\[(\text{define} \ height \ 22)\]

in your program? Answer: it’ll generate an error (at least in the “Advanced Student Language” version of Racket, which is the one we’ll mostly use).

For those who have programmed before, this can seem baffling; they’re used to changing the values associated to ‘variables’ all the time. Don’t worry; we’ll get to a point where we can do something like that. But we absolutely may not redefine things.
By the way, this error is *semantic*, i.e., it has to do with the *meaning* of the program rather than its superficial structure. At the “grammar” (or syntax) level, the program is fine: it consists of two definitions followed by no expression, and that fits the requirement that it be a program.

By analogy, we might say that ”Colorless milk flies furiously” is grammatically correct (”Colorless milk” is the subject, a noun-phrase, and “flies furiously”) is the predicate, consisting of a verb and an adverb. But without a lot of context (or even with it!) it really just makes no sense at all.

**Abstraction**

In using BNF to describe the grammar of Racket, we’ve been applying a general approach: abstraction. That’s the process of looking at a problem and seeing what parts of it are important and what parts are just fluff. As we look at the text of a Racket program, the arrangement of parentheses matters; the particular font we use to display the text does not. In much the same way, the fact that two items are separated by whitespace matters, but how much whitespace does not matter.

Successful abstraction can be incredibly useful, for the solutions to abstracted problems can be applied over and over; failed abstraction can be a mess, as you try to solve a problem that’s so far removed from the original that it’s too tough to address.

**2 Eliza**

The process we’ve used to describe the syntax of Racket — breaking programs into tokens, and seeing whether the resulting sequence of tokens matches up nicely with our BNF description of what constitutes a program — can be used in other contexts as well.

In 1966, Joseph Weizenbaum developed a program called “ELIZA”. It was intended to be a parody of a Rogerian psychotherapist, whose technique is to reflect patients' sentences back at them. When ELIZA was run, it requested an input from a human user, who would type English sentences into the terminal. ELIZA then responded to this input in a seemingly intelligent manner. The Eliza demo represented the TA version of Eliza. When it runs, the program is simply looking for certain patterns in the user’s input. There is always a catch all when it doesn’t match any pattern, which in our case was ”Tell me more”.

There are consequences for every program. There are a lot of people who have access to the internet. Imagine if Eliza is put up online and a user is unaware that Eliza is not a human but in fact a computer program. If the user is in an unstable mental state, they may type in expecting something but eliza’s response might not be up to their expectations. This is definitely not healthy, but what should Eliza return? What can we as programmers do to avoid such situations? One way is to specify that Eliza is NOT a human and is not a professional. Can you think of other ways? Can you think of other repercussions?

**3 Sets**

We now need to talk about functions, but to do so, I have to first talk about *sets*. A set is the mathematical representation of a collection of things. If you have a set $A$, and some object $x$, then
either $x$ is in $A$ or $x$ is not in $A$. For example, Neil Armstrong is not in the set of CS17 students. Items that are in a set $A$ are called the elements of $A$. If $x$ is an element of $A$, we write this in mathematical shorthand by writing

$$x \in A.$$  

If $x$ is not in $A$, we write

$$x \notin A.$$  

There are three main ways of describing sets: natural language descriptions, enumeration, and restriction. We’ll address these one at a time. But first, elements give permanent names to two particular sets.

1. The set $\mathbb{N}$ of natural numbers, consisting of the numbers 0, 1, 2,…. (Some authors prefer to start at 1, but in CS17, the natural numbers start at zero.)

2. The set $\mathbb{R}$ of real numbers, which contains not only integers but also rational numbers like $\frac{2}{3}$, and decimals like $-3.162$ and even irrational numbers like $\pi$.

### 3.1 Natural language descriptions of sets

This amounts to describing, in plain language, what’s in a set. Our description of the natural numbers just above is a good example of a natural-number description.

These are often easy to express, but they tend to take up a fair amount of space. They are also fraught with peril, from the point of view of mathematical logic. (The particular perils will never arise in CS17, fortunately. But they’re one reason why mathematicians tend to shy away from natural-language descriptions.)

### 3.2 Enumeration of sets

*Enumeration* is a way of specifying the members of a finite set: you just list its members, separated by commas, between braces, like this: $S = \{1, 2, 5\}$. That tells you, when you see it, that $1 \in S, 5 \in S, 2 \in S$ are all true statements, that $7 \in S$ is false, and that if I tell you that $x \in S$, you can conclude that $x$ is either 1, 2, or 5.

It’s perfectly OK for an enumeration to be redundant. I could have specified the same set $S$ by writing that $S = \{1, 2, 2, 2, 2, 1, 1, 5\}$, although basically no one ever does so.

### 3.3 Restriction of sets

A common way to describe a set is by saying it consists of all elements of some other set that have some property. For instance, we might write “Let $P$ be the set of all numbers in $\mathbb{R}$ that are positive.” A mathematician would write that like this:

$$P = \{x \in \mathbb{R} \mid x > 0\}.$$  

The vertical bar in that notation is read “such that”.

3
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback)