

Numbers in Scheme

While Scheme excels at symbolic and list processing, it is also quite capable of doing numeric computation. Scheme provides a variety of procedures for dealing with a variety of categories of numbers.

Procedures covered in this reading: `complex?`, `exact?`, `exact->inexact`, `inexact?`, `integer?`, `number?`, `rational?`, and `real?`.

Categories of Numbers

Scheme treats numbers slightly differently depending on whether they are *integers* (whole numbers), *rational numbers* (expressible as a ratio of integers), *real numbers* (corresponding to points on a number line), or *complex numbers* (corresponding to points on the plane determined by a real-number line and a perpendicular line for “imaginary numbers” -- the square roots of negative numbers). From the Scheme programmer’s point of view, these categories of numbers are nested: all integers also qualify as rational numbers (5 is the same thing as 5/1); all rationals count as real numbers, and all real numbers as complex numbers. But, mathematically speaking, the converse inclusions do not generally hold. (3/4 is rational but not an integer, the square root of 2 is real but not rational, and the square root of -1 is complex but not real.)

Scheme supplies a predicate for each of these categories of numbers: `integer?`, `rational?`, `real?`, and `complex?`.

Within each of these categories of numbers, Scheme distinguishes between *exact numbers*, which are guaranteed to be calculated and stored internally with complete accuracy (no rounding off), and *approximations*, also called *inexact numbers*, which are stored internally in a form that conserves the computer’s memory and permits faster computations, but allows small inaccuracies (and occasionally ones that are not so small) to creep in. Since there’s no great advantage in obtaining an answer quickly if it may be incorrect, we shall avoid using approximations in this course, except when the data for our problems are themselves obtained by inexact processes of measurement.

To determine whether Scheme is representing a particular number exactly or inexactly, use one of the predicates `exact?` and `inexact?`.

```
> (exact? 5/9)
#t
> (exact? 13.2)
#f
> (inexact? 13.2)
#t
```

DrScheme happens to store real numbers in such a way that any real number that can be named or computed also counts as rational. For instance, when DrScheme computes `(sqrt 2)`, the value it returns is an inexact approximation to the correct value, and it turns out that DrScheme uses only rational numbers, even when trying to approximate irrational ones.

The standard language definition for Scheme says that an implementation of the language does not have to support all these categories of numbers; it would be legal, for instance, to leave out complex numbers or to treat all numeric values as inexact. However, most implementations (including DrScheme) support all the kinds of numbers described here.

The built-in Scheme procedure `exact->inexact` takes an exact number as its argument and returns an inexact approximation to it:

```
> (exact->inexact 12/7)
1.7142857142857142
```

Since DrScheme uses fractional notation to print out exact numbers, but renders approximations as decimals, invoking this procedure is a simple way to determine the general format in which results are printed. As we'll see later in the semester, however, there are better ways that give the programmer finer control over the format.

The Scheme standard does not directly support the familiar category of *natural numbers*, but we can think of them as being just the same things as Scheme's exact non-negative integers.

Numerals

When you write a numeral into a Scheme program or type one in as part of a definition or command to the interactive interface, the structure of the numeral you type determines the category of the number represented.

One basic rule is that a numeral that contains a decimal point normally stands for an approximation rather than an exact number. Scheme assumes that you may have rounded off the last decimal place and takes this as implicit permission to use a rounded-off representation. If you want Scheme to interpret the numeral as an exact number, you can either convert it to a fraction -- for instance, changing `1.732` to `1732/1000` -- or attach the *exactness prefix* `#e` at the beginning of the numeral, making it `#e1.732`.

Conversely, a number written such as a sequence of digits (possibly with a sign at the beginning) or as a fraction normally stands for an exact number. If you want an approximation instead, use an equivalent numeral with a decimal point or attach the *inexactness prefix* `#i`. (So `23/70` is an exact number, but `#i23/70` is an approximation.)

Scheme permits the use of a version of *scientific notation*, in which a real number is expressed as the product of some coefficient and some integer power of 10. For instance, the numeral `3.17e8` denotes the real number three hundred and seventeen million -- that is, 3.17 times ten to the eighth power. The part of the numeral that precedes the `e` is the coefficient; the part that follows indicates the power of ten by which the coefficient should be multiplied. A number expressed in scientific notation is also inexact unless preceded by the exactness prefix. Chez Scheme uses scientific notation when printing out an inexact number if its absolute value is either very large or very small.