Numbers

Numbers represent numerical quantities. There are two kinds of numbers: standard and extended. Standard numbers have six digits of precision and a range of $\pm 10^{\pm 30}$ and are very efficient in both time and memory. Integers in the range ±32,000 are handled very efficiently and quickly. Standard numbers are what you get by default.

Extended numbers have fifteen digits of accuracy and a range of $\pm 10^{\pm 32,000}$. However, they are about two to three times slower than standard numbers and take much more space.

Both kinds of number represent base-ten quantities exactly with no rounding errors.

In this chapter's examples, x, y, and z represent any numeric values, n and b integer values, and s a string.

Arithmetic

Numbers can perform the standard arithmetic operations and mathematical functions:

x + y
add
x - y
subtract
- x

negate
x * y
multiply
x / y
divide, dividing by zero results in ±infinity
x % y
remainder
x div y
integer division, integer result, rounded towards zero
x.abs
absolute value
x.sqrt
square root
x.sin
trigonometric sine
x.cos
trigonometric cosine
x.tan
trigonometric tangent
x min y

minimum of x and y
x max y
maximum of x and y
n.fib
the n-th Fibonacci number

Comparison

Numbers can be compared with the normal comparison operations. These comparisons all return either true or false.

x == y
equality
x != y
inequality
x < y
x less than y
x <= y
x less than or equal to y
x > y
x greater than y
x >= y
x greater than or equal to y
x between y and z
y <= x, and x <= z

Precedence

The mathematical operators have precedence. This means that a sequence of operations are evaluated as you would evaluate them yourself. The precise rule is based on the concept of precedence. Each operator has a precedence. Operators with high precedence are evaluated before any operators of lower precedence. Operators of the same precedence are evaluated left to right. Of course, you can use parentheses to override this: expressions in parentheses are always evaluated first.

- x <-- Highest Precedence

x * y x / y x % y

x + y x - y

x < y x <= y x >= y x > y
x == y x != y <-- Lowest Precedence

Entering and Displaying

Numbers can be entered in scripts. They follow the standard convention for entering numbers in computers. In addition you can enter integers in base 2, 8 or 16. Some examples are:

165
an integer

3.14159
a number with a decimal part

0.2
just a decimal part

.2
the leading zero is optional (but recommended)

-10
a negative number

5e2
exponent notation: this equals 5 x 102, or 500

0.625e-9
this equals 0.625 x 10-9, or 0.000000000625

0xA5
hexadecimal (base 16) number, this equals 165

0o245
octal (base 8) number, this equals 165

0b10100101
binary (base 2) number, this equal 165

When numbers are displayed, they are always displayed in base 10. In general they are limited to three decimal places and only used exponent notation if needed.

n.name
a string that is the default display format of a number
format n using s
a string of the number formatted according to s

The format string follows rules similar used in most spread sheets: ‘9’s indicated required digit positions, ‘#’s indicate optional. Most other characters are simply copied. Note that it is possible to format numbers in ways that they can’t be entered. This is the same message that number fields use to format their values for display. The number format option of number fields lets you specify the format string.

Some example formats and how they format the numbers: .314159, -23.4, and 181282:

"$#,##9.99"

$0.31
the dollar sign is put in front

-$23.40
forced to two decimal places

$181,282.00
commas added

"9.9###"
0.3142
rounded to four places

-23.4
only needed one decimal place

181282.0
always adds places to the left if needed

".999e"
.314e0
forced exponent notation

-.234e2

.181e6

"#9.9e?"
0.3
exponent notation only if needed

-23.4

1.8e5
too big to fit, so uses exponent

"9%"
31%
percent multiplies the value by 100

-2340%

18128200%

"#,##9.99 cr; #,##9.99 db"
semicolon separates two formats:

0.31 cr
positive numbers use first

23.40 db
negative numbers use second

181,282.00 cr

```
number scan s
```

```
convert a string to a number
number scan-integer s base b

convert a string to n integer
```

These two messages allow you to convert a string to a number. Scan converts decimal numbers with optional decimal points, exponent notation, percent signs and minus signs. It will also interpret a number in parentheses as a negative number (accounting style). Scan-integer will convert only integer strings, but will do so in a number of bases: If the base argument is supplied, then it is used as the base of the number. The base can be anywhere from 2 to 16. If the base isn't specified, then it is assumed to be decimal, unless the string starts with "0x", "0o", or "0b", in which case it is hexadecimal, octal or binary respectively.

!!
Notice that these to messages are sent to the object number, not to any particular number.

Rounding and Truncation

Numbers can be rounded and truncated.

```
round x to n
round x to the n-th power of 10
truncate x to n
truncate x to the n-th power of 10
```

N defaults to 0, which rounds or truncates to integers. If n is -2, then this gives two decimal places, if n is 3 then this rounds or truncates to thousands:

```
 round 2.345 to -2
fi 2.35
 truncate 2.345 to -2
fi 2.34
 round 4590 to 3
fi 5000

truncate 4590 to 3
```

x.integer
integer portion of x (truncates toward zero)
x.fraction
fractional part of x

x is equal to x.integer plus x.fraction
x.ceiling
nearest integer >= to x (truncates toward +∞)
x.floor
nearest integer <= to x (truncates toward -∞)
x.mantissa
mantissa of x, always a number between -1 and 1
x.exponent
exponent of x

x is equal to x.mantissa times 10 raised to x.exponent

Extended Numbers & Conversion

Any numeric operation involving an extended number, results in an extended number. Since extended numbers are less efficient in both time and memory, you will want to use them only when you need the precision, and convert back to standard when you can. (Integer, ceiling, floor, and exponent are exceptions: they produce standard numbers if possible even if their argument is extended).

x.extended
returns an extended number equal to x
x.standard
returns a standard number equal to x

These return an extended or standard version of x respectively. If the number is already in the right format, it just returns it immediately. When converting to standard from extended, the value is rounded as needed.

x.precision
returns the number of decimal digits of precision.

If x is a standard number, then 6. If x is an extended number, then 16. [This is a bug, it should return 15.]

Random Numbers

choose n
returns a random integer between 1 and n, inclusive.
choose n to m
returns a random integer between n and m, inclusive.

number.seed := x.
reset the random seed to x

This resets the random number generator to a particular point, x. X should be a number between zero and one. If you reset the random number generator to the same value, then the same numbers will be returned from the choose message. This is useful if you are debugging a program that uses random numbers, because you can force the system to repeat exactly the same set of choices again.