Aside from the modifications to J's primitive verbs, most of the functionality of JtoLaTeX is kept in scripts located in the `extra` folder. Through some trickery with `Public_j_`, these may be accessed just like a normal addon (see, for instance, the `require 'array'` statement at the top of this document's source). However, functions in the `core` addon are loaded automatically.

1 Low-level verbs

These verbs should be avoided whenever possible. They deal with the string representation of a noun, not its actual structure. Using higher-level verbs instead means your code will adapt sensibly to different arguments; for instance parentheses will be resized with `\left` and `\right` when surrounding a tall expression.

`toString` converts a LaTeX noun to a string, and its inverse, `toL`, converts it back. `toStrings` converts each atom of its argument to a boxed string, and has no inverse.

```
(\\rightarrow |.&.toString)  ='+/ *: c,a,b
```

$$c^2 = a^2 + b^2 \rightarrow 2^b + 2^a = 2^c$$

`concat` combines the left and right arguments, with no intervening space.

```
\_var concat phi
```

$$\varphi$$

`infix` is an adverb that concatenates `x` and `y` with `u` in the middle.

```
\_Long 'right'infix arrow
```

$$\Longrightarrow$$

2 Defining things

A few verbs are included to make the assignments at the beginning of your document a bit easier (and more LaTeX-like).

`is` takes a name on the left and a value on the right, assigns the given value to the name, and returns `i.0 0`.

`declare` is quite a helpful adverb, and a strong candidate for the most complicated line of code I've ever written. Given a string `u` which contains `y`, `u declare` will produce a verb—call it `D`—which runs its right argument through this string as `y` and assigns the result to the left argument. Each argument to `D` must be either an array of boxes or a string, which will be turned into an array with `(;:)`. If the left argument is omitted, the right is used for both name and value.

```
DeclareFunc 'textbf'  NB. I'll get to this later...
DeclareBold =: 'textbf y' declare
DeclareBold 'one two'
'One Two' DeclareBold 'eleven twelve'
list one,two,One,Two
```

one, **two**, **eleven**, **twelve**

Three `declare`-style functions are provided in `core`. Each works like a particular `\` construct from JtoLaTeX's syntax.

- `DeclareConst`: like `\_const`
- `DeclareFunc`: like `\func`
- `DeclareOp`: like `\\op`

For instance, the `DeclareFunc 'textbf'` call above makes `'textbf'` into a function that applies with `{}`. Additionally, `DeclareInfix` declares as an infix operator using `infix`.

3 Environments

The `inenv` verb is a straightforward way to use LaTeX environments:

```
'verbatim' inenv ":i.3


0 1 2
```

To include arguments, append a box to the left argument of `inenv` which gives the string of arguments. You must include the brackets.

```
fig =: (\caption 'a variable.') concat~ mathdisp x
('figure';'[ht]') inenv fig
```

$$x$$

Figure 1: a variable.

The verbs `mathinline` and `mathdisp` place their arguments in inline and display math mode (with `$$` and `\[\]`), respectively.