Syntax of Eigenlanguage

Application

$f\ \ a$

Applies the function f to its argument a.

Singleton

$()$

Represents the only value of a unit type.

Left-Recursive Group

$(x_1\ \ \cdots)$

Builds pairs of expressions x from right to left. For example $(x_1\ x_2\ x_3)$ yields the code $((()\ \ x_1)\ x_2)\ x_3$.

Right-Recursive Group

$[x_1\ \ \cdots]$

Builds pairs of expressions x from left to right. For example $[x_1\ x_2\ x_3]$ yields the code $x_1\ (x_2\ (x_3\ ()))$.

Code as Data

$\backslash x$

Treats the expression x as a value. Works recursively if repeated. For example $\backslash(x_1\ (f\ a_1\ a_2)\ x_3)$ produces the data $(x_1\ y\ x_3)$.

Data as Code

$/x$

Treats the value x as an expression. For example $\backslash\backslash(x_1\ /(f\ a_1\ a_2)\ x_3)$ produces the data $(x_1\ y\ x_3)$.

Binding

$$
\begin{array}{l}
=\ (y_1\ \ x_1 \\
\quad \cdots\ \ \cdots) \\
\ \ z
\end{array}
$$

Binds symbols y to expressions x inside expression z and every x.

Function

$\rightarrow\ \ p\ \ z$

Defines an anonymous function and binds its parameter p to its argument inside z.

Nested Functions

$\rightarrow\ \ (p_1\ \ \cdots)\ \ z$

Defines an anonymous function and binds its parameters p to its arguments inside z. For example $\rightarrow\ (p_1\ p_2\ p_3)\ z$ is equivalent to $\rightarrow\ p_1\ (\rightarrow\ p_2\ (\rightarrow\ p_3\ z))$.

Name Qualification

m/e

Resolves to the exported symbol e from module m.

Module

$$
\begin{array}{l}
\longleftrightarrow\ (m\ \ p_1\ \ \cdots)\ \ (\\
\rightarrow\ \ (e_1 \\
\quad\ \ (=\ (b_2\ \ e_2 \\
\quad\quad\quad\ \ b_3\ \ e_3 \\
\quad\quad\quad\ \ \cdots\ \ \cdots) \\
\quad\quad\ \ \cdots\ \ \cdots) \\
\quad\ \ \cdots) \\
\leftarrow\ \ (i_1 \\
\quad\ \ (i_2\ \ a_{2,1}\ \ \cdots) \\
\quad\ \ (=\ (c_3\ \ i_3 \\
\quad\quad\quad\ \ c_4\ \ (i_4\ \ a_{4,1}\ \ \cdots) \\
\quad\quad\quad\ \ \cdots\ \ \cdots) \\
\quad\quad\ \ \cdots\ \ \cdots) \\
\quad\ \ \cdots) \\
y_1\ \ x_1 \\
\cdots\ \ \cdots \\
)
\end{array}
$$

Declares the module m with parameters p. Exports symbols e, imports modules i and binds symbols y to expressions x inside every e, every a and every x. Gives some exports e aliases b, some imports i aliases c and some imports i arguments a.

Number

```
+18_12
```

Represents the number 20, which is 18 in base 12.

Character

```
'T'
```

Is the 20th character of the alphabet.

String

```
"This text
is arbitrary."
```

Contains text with escape sequences.

External String

```
← "arbitrary_file.text"
```

Reads a file that contains text without escape sequences.

Syntactic Comment

```
%arbitrary-expression
```

Line Comment

```
% This text is arbitrary.
```

Block Comment

```
%%
This text
is arbitrary.
%%
```