The Biscuit Programming Language

A Biscuit program is essentially a list of simple instructions, each terminated by a line break. Each sentence is made of a simple combination of the languages four core atoms:

Keywords	are function names. Some, like **PRINT** or **ADD** expect to be supplied with arguments.
Decimal Literals	represent floating point values.
String Literals	represent values of type string.
Identifiers	refer to variables.

Instructions always start with one keyword, followed by arguments in the form of literals or variable identifiers, if so required by the keyword. A small example program:

```
PRINT     “Hello world!”
ASSIGN    12          my_number
ADD       5           my_number       my_number
PRINT     “This should be 15: ”       my_number
EXIT
```

All **variables** are duck typed, (either string or float) and referred to through identifiers. They do not need to be explicitly declared. A variable is automatically declared once a value is assigned to it, by whichever keyword. Trying to read from a variable which hasn't previously been assigned to, however, will result in an error.

The language has the following built-in **keywords**:

PRINT <any> ...
Prints the supplied list of arguments, including a line break at the end.

ASSIGN <any> <identifier>
Copies the value of the first operand to the variable specified in the second argument.

ADD <number> <number> <identifier>
The values of the first two arguments are added and the result is assigned to the variable in argument three.

SUB <number> <number> <identifier>

The first argument is subtracted by the second and the difference gets assigned to the variable in argument three.

MUL <number> <number> <identifier>
The values of the first two arguments are multiplied and the result is assigned to the variable in argument three.

DIV <number> <number> <identifier>
The first argument is divided by the second and the result gets assigned to the variable in argument three.

EXIT
Exits the program. Always expected at the end of a program.

GOTO <number> <number>
Jumps to the instruction at the index specified by the first argument if the second argument is greater than zero.

JUMP <number> <number>
Skips n instructions as specified in the first argument, if the second argument is greater than zero. Negative values to argument one means jumping backwards.

STRIN <string> <identifier>
Print the supplied prompt string, then takes in a string from the terminal and assigns it to the specified variable.

NUMIN <string> <identifier>
Print the supplied prompt string, then takes in a number from the terminal and assigns it to the specified variable.

RAND <number> <identifier>
Generates an integral number in the range $[0,\ arg_1)$ and assigns it to the specified variable.

EQUAL <number> <number> <identifier>
Assigns 1 to the third argument, if the first two arguments are equal. Else assigns 0.