Lexical Rules

1. Identifiers may be any Unicode alphanumeric character or '`_`' optionally followed by more alphanumeric characters, '`_`', or digits
2. Line-comments start with '`//`' and go to the end of the line
3. Block-comments start with '`/*`' and go until a matching '`*/`'. They may nest.
4. Strings are surrounded by '`"`'. They may contain the following escape sequences
   (a) '`\n`' : character newline (ASCII code point 10)
   (b) '`\r`' : carriage return (ASCII code point 13)
   (c) '`\t`' : horizontal tab (ASCII code point 9)
   (d) '`\"`' : literal " (ASCII code point 34)
   (e) TODO: add Unicode escape sequences
5. Format strings are start with by '`f"`' and end with '`"`'. They may contain the same escape sequences as normal strings. Additionally, they may contain expressions that are surrounded by '`{`' and '`}`'. Due to current restrictions, these expressions cannot contain '`{`' or '`}`'.

Grammar

In the following grammar specification, ⟨UPPERCASE⟩ denotes non-terminal rules, whereas ⟨lowercase⟩ denotes a token. '`literal`' denotes a simple token

Every comma separated list may be empty. Trailing commas are not yet supported.

⟨*PROGRAM*⟩ ::= (⟨*ITEM*⟩)*

⟨*ITEM*⟩ ::= ⟨*FUNCTION*⟩

⟨*FUNCTION*⟩ ::= '`toaster`' ⟨*ident*⟩ ⟨*HAPPINESS*⟩
⟨*ARG_LIST*⟩ ⟨*RETURN_TYPE*⟩
⟨*BLOCK*⟩

⟨*HAPPINESS*⟩ ::= '`:>`'
| '`:<`'

⟨*ARG_LIST*⟩ ::= ⟨*ident*⟩$_1$ '`:`' ⟨*TYPE*⟩$_1$ '`,`' ... '`,`' ⟨*ident*⟩$_n$ '`:`' ⟨*TYPE*⟩$_n$

⟨*RETURN_TYPE*⟩ ::= '`->`' ⟨*TYPE*⟩
| ε

⟨*TYPE*⟩ ::= ⟨*ident*⟩
| '`(`' ⟨*TYPE*⟩ '`)`'
| '`(`' ⟨*TYPE*⟩$_1$ '`,`' ... '`,`' ⟨*TYPE*⟩$_n$ '`)`'

⟨*BLOCK*⟩ ::= '`{`' (⟨*STATEMENT*⟩)* ⟨*EXPR*⟩? '`}`'

⟨*STATEMENT*⟩ ::= ‘`return`’ ‘`;`’
| ‘`return`’ ⟨*EXPR*⟩ ‘`;`’
| ‘`let`’ ⟨*PATTERN*⟩ ‘`=`’ ⟨*EXPR*⟩ ‘`;`’
| ⟨*IF_EXPR*⟩
| ⟨*RHS*⟩ ‘`=`’ ⟨*EXPR*⟩ ‘`;`’

⟨*EXPR*⟩ ::= ⟨*integer*⟩
| ⟨*string*⟩
| ⟨*EXPR*⟩ ⟨*BINOP*⟩ ⟨*EXPR*⟩
| ⟨*UNOP*⟩ ⟨*EXPR*⟩
| ⟨*FUNC_CALL*⟩
| ‘`(`’ ⟨*EXPR*⟩ ‘`)`’
| ⟨*IF_EXPR*⟩

⟨*IF_EXPR*⟩ ::= ‘`if`’ ⟨*EXPR*⟩ ⟨*BLOCK*⟩
(‘`else`’ ‘`if`’ ⟨*EXPR*⟩ ⟨*BLOCK*⟩)*
(‘`else`’ ⟨*BLOCK*⟩)?