- **newline**:
  Newlines are combined, so a single newline is the same as multiple.

- **module-name**:
  Modules are flags given to the interpreter/compiler, to let it know you want to be using certain rules, functions, or features. Available modules: `FORS` (randomness), `FRACTIO` (fractions), `MAGNVM` (large integers), `SCRIPTA` (file I/O: `LEGE`, `SCRIBE`, `ADIVNGE`), `SVBNVLLA` (negative literals).

- **id**:
  Variable. Can only consist of lowercase characters and underscores, but not the letters j, u, or w.

- **builtin**:
  Builtin functions are uppercase latin words.

- **string**:
  Any text encased in `"` or `'` characters. Single-quoted strings are always literal. Strings support 1-based indexing (`string[I]`) and inclusive slicing (`string[I VSQVE III]`), returning single-character strings and substrings respectively.

- **interpolated-string**:
  A double-quoted string containing `{`*expression*`}` segments. Each expression is evaluated and coerced to a string. Use `{{` and `}}` for literal braces.

- **numeral**:
  Roman numerals consisting of the uppercase characters I, V, X, L, C, D, and M. Can also include underscore if the module MAGNVM.

- **bool**:
  VERITAS or FALSITAS.

- **binop**:
  Binary operators: `+`, `-`, `*`, `/`, `RELIQVVM` (modulo), `EST` (equality), `DISPAR` (not-equal), `MINVS` (<), `PLVS` (>), `ET` (and), `AVT` (or), `&` (string concatenation).

- **unop**:
  Unary operators: `-` (negation), `NON` (boolean not).

Top-level		
program	→	*optional-newline module-calls statements*
module-calls	→	*module-call* **newline** *module-calls*
module-calls	→	
module-call	→	`CUM` **module-name**
statements	→	*statement* **newline** *statements*
statements	→	
optional-newline	→	**newline**
optional-newline	→	

Statements		
statement	→	*expression*
statement	→	`DESIGNA` **id** `VT` *expression*
statement	→	`DESIGNA` **id** `,` *ids* `VT` *expression*
statement	→	**id** `AVGE` *expression*
statement	→	**id** `MINVE` *expression*
statement	→	`DEFINI` **id** `(` *optional-ids* `)` `VT` *scope*
statement	→	*if-statement*
statement	→	`DVM` *expression* `FAC` *scope*
statement	→	`AETERNVM FAC` *scope*
statement	→	`PER` **id** `IN` *expression* `FAC` *scope*
statement	→	`DONICVM` **id** `VT` *expression* `VSQVE` *expression* `FAC` *scope*
statement	→	`REDI(` *optional-expressions* `)`
statement	→	`ERVMPE`
statement	→	`CONTINVA`
statement	→	*try-statement*
try-statement	→	`TEMPTA` *scope* `CAPE` **id** *scope*
if-statement	→	`SI` *expression* `TVNC` *scope*
if-statement	→	`SI` *expression* `TVNC` *scope optional-newline else-statement*
else-statement	→	`ALIVD` *scope*
else-statement	→	`ALIVD` *if-statement*
scope	→	*optional-newline* `{` **newline** *statements* `}`

Expressions		
expression	→	`(` *expression* `)`
expression	→	**id**
expression	→	**builtin** `(` *optional-expressions* `)`
expression	→	`INVOCA` *expression* `(` *optional-expressions* `)`
expression	→	`FVNCTIO (` *optional-ids* `)` `VT` *scope*
expression	→	*literal*
expression	→	*expression* `[` *expression* `]`
expression	→	*expression* `[` *expression* `VSQVE` *expression* `]` (inclusive slice)
expression	→	*expression* **binop** *expression*
expression	→	**unop** *expression*
literal	→	**string**
literal	→	**interpolated-string**
literal	→	**numeral**
literal	→	**bool**
literal	→	`[` *optional-expressions* `]`
literal	→	`[` *expression* `VSQVE` *expression* `]` (inclusive on both ends)
literal	→	`TABVLA {` *optional-dict-items* `}`
optional-dict-items	→	*dict-items*
optional-dict-items	→	
dict-items	→	*expression* `VT` *expression* `,` *dict-items*
dict-items	→	*expression* `VT` *expression*

Lists		
optional-ids	→	ids
optional-ids	→	
ids	→	**id**, *ids*
ids	→	**id**
optional-expressions	→	expressions
optional-expressions	→	
expressions	→	*expression*, *expressions*
expressions	→	*expression*