DESIGN and DCG for TATASCRIPT

DCG:

```
:- table condition/2.
% Entry point for program
program --> [;], main_block, [;].
main_block --> statements.

% Multiple statements separated by periods
statements --> statement, statements.
statements --> statement.

% Different types of statements
statement --> assignment, [.].
statement --> display, [.].
statement --> if_statement.
statement --> for_loop.
statement --> while_loop.
statement --> increment, [.].

% Assignment statement
assignment --> data_type, variable, [=], expression.

assignment --> variable, [=], expression.

% Display statement
display --> [display], [->], output.

output --> string_literal | variable | number.

% If-Else Statements
if_statement --> [if, ->], condition, block, elseif_blocks, else_block.

% Ternary Statements
ternary_statement --> condition, [->], ternary_expression, [<-], ternary_expression.
ternary_expression --> expression | statements.

% Optional Else-If and Else blocks
elseif_blocks --> elseif_block, elseif_blocks.
elseif_blocks --> [].

elseif_block --> [<-, ->], condition, block.

else_block --> [<-], block.
else_block --> [].

% For Loop
for_loop --> [for, ->], init, [:], condition, [:], increment, block.

% While Loop
while_loop --> [while, ->], condition, block.
```

```
% Code block
block --> ['{'], statements, ['}'].

block --> ['{'],['}'].

% Expressions
expression --> term.
expression --> term, arithmetic_op, expression.
expression --> increment.
expression --> ternary_statement.

term --> variable.
term --> number.
term --> string_literal.
term --> expression.

% Conditions
condition --> term, relational_op, term.
condition --> term, boolean_op, condition.
condition --> condition, logical_op, condition.

% Initialization in for loop
init --> assignment.

% Increment in for loop
increment --> variable, increment_op.

% Operators
arithmetic_op --> [+] | [-] | [*] | [/].
relational_op --> [<] | [>] | [==] | [<=] | [>=] | [!].
boolean_op --> [true] | [false].
increment_op --> [++] | [--].
logical_op --> [&] | ['|'].

% Data Types
data_type --> [int] | [string] | [bool].

% Variable names
variable --> [Var], { atom(Var), atom_chars(Var, [FirstChar | Rest]), char_type(FirstChar,
alpha), all_alnum_or_underscore(Rest) }.

% Data types
number --> [Num], { number(Num) }.
string_literal --> [Str], { atom(Str), atom_concat("", _, Str), atom_concat(_, "", Str) }.

% Helper predicate for valid variable names
all_alnum_or_underscore([]).
all_alnum_or_underscore([Char | Rest]) :-
    (char_type(Char, alnum) ; Char == '_'),
    all_alnum_or_underscore(Rest).
```