**Solidity BNF:**

SourceUnit = (PragmaDirective | ImportDirective | ContractDefinition)\*

// Pragma actually parses anything up to the trailing ';' to be fully forward-compatible.

PragmaDirective = 'pragma' Identifier ([^;]+) ';'

ImportDirective = 'import' StringLiteral ('as' Identifier)? ';'

| 'import' ('\*' | Identifier) ('as' Identifier)? 'from' StringLiteral ';'

| 'import' '{' Identifier ('as' Identifier)? ( ',' Identifier ('as' Identifier)? )\* '}' 'from' StringLiteral ';'

ContractDefinition = ( 'contract' | 'library' | 'interface' ) Identifier

( 'is' InheritanceSpecifier (',' InheritanceSpecifier )\* )?

'{' ContractPart\* '}'

ContractPart = StateVariableDeclaration | UsingForDeclaration

| StructDefinition | ModifierDefinition | FunctionDefinition | EventDefinition | EnumDefinition

InheritanceSpecifier = UserDefinedTypeName ( '(' Expression ( ',' Expression )\* ')' )?

StateVariableDeclaration = TypeName ( 'public' | 'internal' | 'private' | 'constant' )? Identifier ('=' Expression)? ';'

UsingForDeclaration = 'using' Identifier 'for' ('\*' | TypeName) ';'

StructDefinition = 'struct' Identifier '{'

( VariableDeclaration ';' (VariableDeclaration ';')\* )? '}'

ModifierDefinition = 'modifier' Identifier ParameterList? Block

ModifierInvocation = Identifier ( '(' ExpressionList? ')' )?

FunctionDefinition = 'function' Identifier? ParameterList

( ModifierInvocation | StateMutability | 'external' | 'public' | 'internal' | 'private' )\*

( 'returns' ParameterList )? ( ';' | Block )

EventDefinition = 'event' Identifier IndexedParameterList 'anonymous'? ';'

EnumValue = Identifier

EnumDefinition = 'enum' Identifier '{' EnumValue? (',' EnumValue)\* '}'

IndexedParameterList = '(' ( TypeName 'indexed'? Identifier? (',' TypeName 'indexed'? Identifier?)\* )? ')'

ParameterList = '(' ( TypeName Identifier? (',' TypeName Identifier?)\* )? ')'

TypeNameList = '(' ( TypeName (',' TypeName )\* )? ')'

// semantic restriction: mappings and structs (recursively) containing mappings

// are not allowed in argument lists

VariableDeclaration = TypeName StorageLocation? Identifier

TypeName = ElementaryTypeName

| UserDefinedTypeName

| Mapping

| ArrayTypeName

| FunctionTypeName

UserDefinedTypeName = Identifier ( '.' Identifier )\*

Mapping = 'mapping' '(' ElementaryTypeName '=>' TypeName ')'

ArrayTypeName = TypeName '[' Expression? ']'

FunctionTypeName = 'function' TypeNameList ( 'internal' | 'external' | StateMutability )\*

( 'returns' TypeNameList )?

StorageLocation = 'memory' | 'storage'

StateMutability = 'pure' | 'constant' | 'view' | 'payable'

Block = '{' Statement\* '}'

Statement = IfStatement | WhileStatement | ForStatement | Block | InlineAssemblyStatement |

( DoWhileStatement | PlaceholderStatement | Continue | Break | Return |

Throw | SimpleStatement ) ';'

ExpressionStatement = Expression

IfStatement = 'if' '(' Expression ')' Statement ( 'else' Statement )?

WhileStatement = 'while' '(' Expression ')' Statement

PlaceholderStatement = '\_'

SimpleStatement = VariableDefinition | ExpressionStatement

ForStatement = 'for' '(' (SimpleStatement)? ';' (Expression)? ';' (ExpressionStatement)? ')' Statement

InlineAssemblyStatement = 'assembly' StringLiteral? InlineAssemblyBlock

DoWhileStatement = 'do' Statement 'while' '(' Expression ')'

Continue = 'continue'

Break = 'break'

Return = 'return' Expression?

Throw = 'throw'

VariableDefinition = ('var' IdentifierList | VariableDeclaration) ( '=' Expression )?

IdentifierList = '(' ( Identifier? ',' )\* Identifier? ')'

// Precedence by order (see github.com/ethereum/solidity/pull/732)

Expression

= Expression ('++' | '--')

| NewExpression

| IndexAccess

| MemberAccess

| FunctionCall

| '(' Expression ')'

| ('!' | '~' | 'delete' | '++' | '--' | '+' | '-') Expression

| Expression '\*\*' Expression

| Expression ('\*' | '/' | '%') Expression

| Expression ('+' | '-') Expression

| Expression ('<<' | '>>') Expression

| Expression '&' Expression

| Expression '^' Expression

| Expression '|' Expression

| Expression ('<' | '>' | '<=' | '>=') Expression

| Expression ('==' | '!=') Expression

| Expression '&&' Expression

| Expression '||' Expression

| Expression '?' Expression ':' Expression

| Expression ('=' | '|=' | '^=' | '&=' | '<<=' | '>>=' | '+=' | '-=' | '\*=' | '/=' | '%=') Expression

| PrimaryExpression

PrimaryExpression = BooleanLiteral

| NumberLiteral

| HexLiteral

| StringLiteral

| TupleExpression

| Identifier

| ElementaryTypeNameExpression

ExpressionList = Expression ( ',' Expression )\*

NameValueList = Identifier ':' Expression ( ',' Identifier ':' Expression )\*

FunctionCall = Expression '(' FunctionCallArguments ')'

FunctionCallArguments = '{' NameValueList? '}'

| ExpressionList?

NewExpression = 'new' TypeName

MemberAccess = Expression '.' Identifier

IndexAccess = Expression '[' Expression? ']'

BooleanLiteral = 'true' | 'false'

NumberLiteral = ( HexNumber | DecimalNumber ) (' ' NumberUnit)?

NumberUnit = 'wei' | 'szabo' | 'finney' | 'ether'

| 'seconds' | 'minutes' | 'hours' | 'days' | 'weeks' | 'years'

HexLiteral = 'hex' ('"' ([0-9a-fA-F]{2})\* '"' | '\'' ([0-9a-fA-F]{2})\* '\'')

StringLiteral = '"' ([^"\r\n\\] | '\\' .)\* '"'

Identifier = [a-zA-Z\_$] [a-zA-Z\_$0-9]\*

HexNumber = '0x' [0-9a-fA-F]+

DecimalNumber = [0-9]+

TupleExpression = '(' ( Expression ( ',' Expression )\* )? ')'

| '[' ( Expression ( ',' Expression )\* )? ']'

ElementaryTypeNameExpression = ElementaryTypeName

ElementaryTypeName = 'address' | 'bool' | 'string' | 'var'

| Int | Uint | Byte | Fixed | Ufixed

Int = 'int' | 'int8' | 'int16' | 'int24' | 'int32' | 'int40' | 'int48' | 'int56' | 'int64' | 'int72' | 'int80' | 'int88' | 'int96' | 'int104' | 'int112' | 'int120' | 'int128' | 'int136' | 'int144' | 'int152' | 'int160' | 'int168' | 'int176' | 'int184' | 'int192' | 'int200' | 'int208' | 'int216' | 'int224' | 'int232' | 'int240' | 'int248' | 'int256'

Uint = 'uint' | 'uint8' | 'uint16' | 'uint24' | 'uint32' | 'uint40' | 'uint48' | 'uint56' | 'uint64' | 'uint72' | 'uint80' | 'uint88' | 'uint96' | 'uint104' | 'uint112' | 'uint120' | 'uint128' | 'uint136' | 'uint144' | 'uint152' | 'uint160' | 'uint168' | 'uint176' | 'uint184' | 'uint192' | 'uint200' | 'uint208' | 'uint216' | 'uint224' | 'uint232' | 'uint240' | 'uint248' | 'uint256'

Byte = 'byte' | 'bytes' | 'bytes1' | 'bytes2' | 'bytes3' | 'bytes4' | 'bytes5' | 'bytes6' | 'bytes7' | 'bytes8' | 'bytes9' | 'bytes10' | 'bytes11' | 'bytes12' | 'bytes13' | 'bytes14' | 'bytes15' | 'bytes16' | 'bytes17' | 'bytes18' | 'bytes19' | 'bytes20' | 'bytes21' | 'bytes22' | 'bytes23' | 'bytes24' | 'bytes25' | 'bytes26' | 'bytes27' | 'bytes28' | 'bytes29' | 'bytes30' | 'bytes31' | 'bytes32'

Fixed = 'fixed' | ( 'fixed' DecimalNumber 'x' DecimalNumber )

Ufixed = 'ufixed' | ( 'ufixed' DecimalNumber 'x' DecimalNumber )

InlineAssemblyBlock = '{' AssemblyItem\* '}'

AssemblyItem = Identifier | FunctionalAssemblyExpression | InlineAssemblyBlock | AssemblyLocalBinding | AssemblyAssignment | AssemblyLabel | NumberLiteral | StringLiteral | HexLiteral

AssemblyLocalBinding = 'let' Identifier ':=' FunctionalAssemblyExpression

AssemblyAssignment = ( Identifier ':=' FunctionalAssemblyExpression ) | ( '=:' Identifier )

AssemblyLabel = Identifier ':'

FunctionalAssemblyExpression = Identifier '(' AssemblyItem? ( ',' AssemblyItem )\* ')'

**JAVASCRIPT BNF:**

[**https://262.ecma-international.org/14.0/#prod-Expression**](https://262.ecma-international.org/14.0/#prod-Expression)

Se debe acceder a la sección “A Grammar Summary”.