**BNF de Javascript**

## **NON-TERMINALS**

| PrimaryExpression | ::= | "this" |
| --- | --- | --- |
|  | | | [ObjectLiteral](https://tomcopeland.blogs.com/EcmaScript.html#prod2) |
|  | | | ( "(" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" )| |
|  | | | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) |
|  | | | [ArrayLiteral](https://tomcopeland.blogs.com/EcmaScript.html#prod5) |
|  | | | [Literal](https://tomcopeland.blogs.com/EcmaScript.html#prod6) |
| Literal | ::= | ( <DECIMAL\_LITERAL> | <HEX\_INTEGER\_LITERAL> | <STRING\_LITERAL> | <BOOLEAN\_LITERAL> | <NULL\_LITERAL> | <REGULAR\_EXPRESSION\_LITERAL> ) |
| Identifier | ::= | <IDENTIFIER\_NAME> |
| ArrayLiteral | ::= | "[" ( ( [Elision](https://tomcopeland.blogs.com/EcmaScript.html#prod7) )? "]" | [ElementList](https://tomcopeland.blogs.com/EcmaScript.html#prod8) [Elision](https://tomcopeland.blogs.com/EcmaScript.html#prod7) "]" | ( [ElementList](https://tomcopeland.blogs.com/EcmaScript.html#prod8) )? "]" ) |
| ElementList | ::= | ( [Elision](https://tomcopeland.blogs.com/EcmaScript.html#prod7) )? [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) ( [Elision](https://tomcopeland.blogs.com/EcmaScript.html#prod7) [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) )\* |
| Elision | ::= | ( "," )+ |
| ObjectLiteral | ::= | "{" ( [PropertyNameAndValueList](https://tomcopeland.blogs.com/EcmaScript.html#prod10) )? "}" |
| PropertyNameAndValueList | ::= | [PropertyNameAndValue](https://tomcopeland.blogs.com/EcmaScript.html#prod11) ( "," [PropertyNameAndValue](https://tomcopeland.blogs.com/EcmaScript.html#prod11) | "," )\* |
| PropertyNameAndValue | ::= | [PropertyName](https://tomcopeland.blogs.com/EcmaScript.html#prod12) ":" [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) |
| PropertyName | ::= | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) |
|  | | | <STRING\_LITERAL> |
|  | | | <DECIMAL\_LITERAL> |
| MemberExpression | ::= | ( ( [FunctionExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod14) | [PrimaryExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod1) ) ( [MemberExpressionPart](https://tomcopeland.blogs.com/EcmaScript.html#prod15) )\* ) |
|  | | | [AllocationExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod16) |
| MemberExpressionForIn | ::= | ( ( [FunctionExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod14) | [PrimaryExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod1) ) ( [MemberExpressionPart](https://tomcopeland.blogs.com/EcmaScript.html#prod15) )\* ) |
| AllocationExpression | ::= | ( "new" [MemberExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod13) ( ( [Arguments](https://tomcopeland.blogs.com/EcmaScript.html#prod18) ( [MemberExpressionPart](https://tomcopeland.blogs.com/EcmaScript.html#prod15) )\* )\* ) ) |
| MemberExpressionPart | ::= | ( "[" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) "]" ) |
|  | | | ( "." [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ) |
| CallExpression | ::= | [MemberExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod13) [Arguments](https://tomcopeland.blogs.com/EcmaScript.html#prod18) ( [CallExpressionPart](https://tomcopeland.blogs.com/EcmaScript.html#prod20) )\* |
| CallExpressionForIn | ::= | [MemberExpressionForIn](https://tomcopeland.blogs.com/EcmaScript.html#prod17) [Arguments](https://tomcopeland.blogs.com/EcmaScript.html#prod18) ( [CallExpressionPart](https://tomcopeland.blogs.com/EcmaScript.html#prod20) )\* |
| CallExpressionPart | ::= | [Arguments](https://tomcopeland.blogs.com/EcmaScript.html#prod18) |
|  | | | ( "[" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) "]" ) |
|  | | | ( "." [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ) |
| Arguments | ::= | "(" ( [ArgumentList](https://tomcopeland.blogs.com/EcmaScript.html#prod22) )? ")" |
| ArgumentList | ::= | [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) ( "," [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) )\* |

| LeftHandSideExpression | ::= | [CallExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod19) |
| --- | --- | --- |
|  | | | [MemberExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod13) |
| LeftHandSideExpressionForIn | ::= | [CallExpressionForIn](https://tomcopeland.blogs.com/EcmaScript.html#prod21) |
|  | | | [MemberExpressionForIn](https://tomcopeland.blogs.com/EcmaScript.html#prod17) |
| PostfixExpression | ::= | [LeftHandSideExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod23) ( [PostfixOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod26) )? |
| PostfixOperator | ::= | ( "++" | "--" ) |
| UnaryExpression | ::= | ( [PostfixExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod25) | ( [UnaryOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod28) [UnaryExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod27) )+ ) |
| UnaryOperator | ::= | ( "delete" | "void" | "typeof" | "++" | "--" | "+" | "-" | "~" | "!" ) |
| MultiplicativeExpression | ::= | [UnaryExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod27) ( [MultiplicativeOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod30) [UnaryExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod27) )\* |
| MultiplicativeOperator | ::= | ( "\*" | <SLASH> | "%" ) |
| AdditiveExpression | ::= | [MultiplicativeExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod29) ( [AdditiveOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod32) [MultiplicativeExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod29) )\* |
| AdditiveOperator | ::= | ( "+" | "-" ) |
| ShiftExpression | ::= | [AdditiveExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod31) ( [ShiftOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod34) [AdditiveExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod31) )\* |
| ShiftOperator | ::= | ( "<<" | ">>" | ">>>" ) |
| RelationalExpression | ::= | [ShiftExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod33) ( [RelationalOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod36) [ShiftExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod33) )\* |
| RelationalOperator | ::= | ( "<" | ">" | "<=" | ">=" | "instanceof" | "in" ) |
| RelationalExpressionNoIn | ::= | [ShiftExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod33) ( [RelationalNoInOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod38) [ShiftExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod33) )\* |
| RelationalNoInOperator | ::= | ( "<" | ">" | "<=" | ">=" | "instanceof" ) |
| EqualityExpression | ::= | [RelationalExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod35) ( [EqualityOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod40) [RelationalExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod35) )\* |
| EqualityExpressionNoIn | ::= | [RelationalExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod37) ( [EqualityOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod40) [RelationalExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod37) )\* |
| EqualityOperator | ::= | ( "==" | "!=" | "===" | "!==" ) |
| BitwiseANDExpression | ::= | [EqualityExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod39) ( [BitwiseANDOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod43) [EqualityExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod39) )\* |
| BitwiseANDExpressionNoIn | ::= | [EqualityExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod41) ( [BitwiseANDOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod43) [EqualityExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod41) )\* |
| BitwiseANDOperator | ::= | "&" |
| BitwiseXORExpression | ::= | [BitwiseANDExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod42) ( [BitwiseXOROperator](https://tomcopeland.blogs.com/EcmaScript.html#prod46) [BitwiseANDExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod42) )\* |
| BitwiseXORExpressionNoIn | ::= | [BitwiseANDExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod44) ( [BitwiseXOROperator](https://tomcopeland.blogs.com/EcmaScript.html#prod46) [BitwiseANDExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod44) )\* |
| BitwiseXOROperator | ::= | "^" |
| BitwiseORExpression | ::= | [BitwiseXORExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod45) ( [BitwiseOROperator](https://tomcopeland.blogs.com/EcmaScript.html#prod49) [BitwiseXORExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod45) )\* |
| BitwiseORExpressionNoIn | ::= | [BitwiseXORExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod47) ( [BitwiseOROperator](https://tomcopeland.blogs.com/EcmaScript.html#prod49) [BitwiseXORExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod47) )\* |
| BitwiseOROperator | ::= | "|" |
| LogicalANDExpression | ::= | [BitwiseORExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod48) ( [LogicalANDOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod52) [BitwiseORExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod48) )\* |
| LogicalANDExpressionNoIn | ::= | [BitwiseORExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod50) ( [LogicalANDOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod52) [BitwiseORExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod50) )\* |
| LogicalANDOperator | ::= | "&&" |
| LogicalORExpression | ::= | [LogicalANDExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod51) ( [LogicalOROperator](https://tomcopeland.blogs.com/EcmaScript.html#prod55) [LogicalANDExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod51) )\* |
| LogicalORExpressionNoIn | ::= | [LogicalANDExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod53) ( [LogicalOROperator](https://tomcopeland.blogs.com/EcmaScript.html#prod55) [LogicalANDExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod53) )\* |
| LogicalOROperator | ::= | "||" |
| ConditionalExpression | ::= | [LogicalORExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod54) ( "?" [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) ":" [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) )? |
| ConditionalExpressionNoIn | ::= | [LogicalORExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod56) ( "?" [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) ":" [AssignmentExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod59) )? |
| AssignmentExpression | ::= | ( [LeftHandSideExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod23) [AssignmentOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod60) [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) | [ConditionalExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod57) ) |
| AssignmentExpressionNoIn | ::= | ( [LeftHandSideExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod23) [AssignmentOperator](https://tomcopeland.blogs.com/EcmaScript.html#prod60) [AssignmentExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod59) | [ConditionalExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod58) ) |
| AssignmentOperator | ::= | ( "=" | "\*=" | <SLASHASSIGN> | "%=" | "+=" | "-=" | "<<=" | ">>=" | ">>>=" | "&=" | "^=" | "|=" ) |
| Expression | ::= | [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) ( "," [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) )\* |
| ExpressionNoIn | ::= | [AssignmentExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod59) ( "," [AssignmentExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod59) )\* |

| Statement | ::= | [Block](https://tomcopeland.blogs.com/EcmaScript.html#prod63) |
| --- | --- | --- |
|  | | | [JScriptVarStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod64) |
|  | | | [VariableStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod65) |
|  | | | [EmptyStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod66) |
|  | | | [LabelledStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod67) |
|  | | | [ExpressionStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod68) |
|  | | | [IfStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod69) |
|  | | | [IterationStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod70) |
|  | | | [ContinueStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod71) |
|  | | | [BreakStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod72) |
|  | | | [ImportStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod73) |
|  | | | [ReturnStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod74) |
|  | | | [WithStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod75) |
|  | | | [SwitchStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod76) |
|  | | | [ThrowStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod77) |
|  | | | [TryStatement](https://tomcopeland.blogs.com/EcmaScript.html#prod78) |
| Block | ::= | "{" ( [StatementList](https://tomcopeland.blogs.com/EcmaScript.html#prod79) )? "}" |
| StatementList | ::= | ( [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) )+ |
| VariableStatement | ::= | "var" [VariableDeclarationList](https://tomcopeland.blogs.com/EcmaScript.html#prod80) ( ";" )? |
| VariableDeclarationList | ::= | [VariableDeclaration](https://tomcopeland.blogs.com/EcmaScript.html#prod81) ( "," [VariableDeclaration](https://tomcopeland.blogs.com/EcmaScript.html#prod81) )\* |
| VariableDeclarationListNoIn | ::= | [VariableDeclarationNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod83) ( "," [VariableDeclarationNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod83) )\* |
| VariableDeclaration | ::= | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ( [Initialiser](https://tomcopeland.blogs.com/EcmaScript.html#prod84) )? |
| VariableDeclarationNoIn | ::= | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ( [InitialiserNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod85) )? |
| Initialiser | ::= | "=" [AssignmentExpression](https://tomcopeland.blogs.com/EcmaScript.html#prod9) |
| InitialiserNoIn | ::= | "=" [AssignmentExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod59) |
| EmptyStatement | ::= | ";" |
| ExpressionStatement | ::= | [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ( ";" )? |
| IfStatement | ::= | "if" "(" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) ( "else" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) )? |
| IterationStatement | ::= | ( "do" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) "while" "(" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" ( ";" )? ) |
|  | | | ( "while" "(" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) ) |
|  | | | ( "for" "(" ( [ExpressionNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod61) )? ";" ( [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) )? ";" ( [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) )? ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) ) |
|  | | | ( "for" "(" "var" [VariableDeclarationList](https://tomcopeland.blogs.com/EcmaScript.html#prod80) ";" ( [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) )? ";" ( [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) )? ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) ) |
|  | | | ( "for" "(" "var" [VariableDeclarationNoIn](https://tomcopeland.blogs.com/EcmaScript.html#prod83) "in" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) ) |
|  | | | ( "for" "(" [LeftHandSideExpressionForIn](https://tomcopeland.blogs.com/EcmaScript.html#prod24) "in" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) ) |
| ContinueStatement | ::= | "continue" ( [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) )? ( ";" )? |
| BreakStatement | ::= | "break" ( [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) )? ( ";" )? |
| ReturnStatement | ::= | "return" ( [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) )? ( ";" )? |
| WithStatement | ::= | "with" "(" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) |
| SwitchStatement | ::= | "switch" "(" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ")" [CaseBlock](https://tomcopeland.blogs.com/EcmaScript.html#prod86) |
| CaseBlock | ::= | "{" ( [CaseClauses](https://tomcopeland.blogs.com/EcmaScript.html#prod87) )? ( "}" | [DefaultClause](https://tomcopeland.blogs.com/EcmaScript.html#prod88) ( [CaseClauses](https://tomcopeland.blogs.com/EcmaScript.html#prod87) )? "}" ) |
| CaseClauses | ::= | ( [CaseClause](https://tomcopeland.blogs.com/EcmaScript.html#prod89) )+ |
| CaseClause | ::= | ( ( "case" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ":" ) ) ( [StatementList](https://tomcopeland.blogs.com/EcmaScript.html#prod79) )? |
| DefaultClause | ::= | ( ( "default" ":" ) ) ( [StatementList](https://tomcopeland.blogs.com/EcmaScript.html#prod79) )? |
| LabelledStatement | ::= | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ":" [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) |
| ThrowStatement | ::= | "throw" [Expression](https://tomcopeland.blogs.com/EcmaScript.html#prod3) ( ";" )? |
| TryStatement | ::= | "try" [Block](https://tomcopeland.blogs.com/EcmaScript.html#prod63) ( ( [Finally](https://tomcopeland.blogs.com/EcmaScript.html#prod90) | [Catch](https://tomcopeland.blogs.com/EcmaScript.html#prod91) ( [Finally](https://tomcopeland.blogs.com/EcmaScript.html#prod90) )? ) ) |
| Catch | ::= | "catch" "(" [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ")" [Block](https://tomcopeland.blogs.com/EcmaScript.html#prod63) |
| Finally | ::= | "finally" [Block](https://tomcopeland.blogs.com/EcmaScript.html#prod63) |

| FunctionDeclaration | ::= | "function" [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ( "(" ( [FormalParameterList](https://tomcopeland.blogs.com/EcmaScript.html#prod93) )? ")" ) [FunctionBody](https://tomcopeland.blogs.com/EcmaScript.html#prod94) |
| --- | --- | --- |
| FunctionExpression | ::= | "function" ( [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) )? ( "(" ( [FormalParameterList](https://tomcopeland.blogs.com/EcmaScript.html#prod93) )? ")" ) [FunctionBody](https://tomcopeland.blogs.com/EcmaScript.html#prod94) |
| FormalParameterList | ::= | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ( "," [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) )\* |
| FunctionBody | ::= | "{" ( [SourceElements](https://tomcopeland.blogs.com/EcmaScript.html#prod95) )? "}" |
| Program | ::= | ( [SourceElements](https://tomcopeland.blogs.com/EcmaScript.html#prod95) )? <EOF> |
| SourceElements | ::= | ( [SourceElement](https://tomcopeland.blogs.com/EcmaScript.html#prod97) )+ |
| SourceElement | ::= | [FunctionDeclaration](https://tomcopeland.blogs.com/EcmaScript.html#prod92) |
|  | | | [Statement](https://tomcopeland.blogs.com/EcmaScript.html#prod62) |
| ImportStatement | ::= | "import" [Name](https://tomcopeland.blogs.com/EcmaScript.html#prod98) ( "." "\*" )? ";" |
| Name | ::= | <IDENTIFIER\_NAME> ( "." <IDENTIFIER\_NAME> )\* |
| JScriptVarStatement | ::= | "var" [JScriptVarDeclarationList](https://tomcopeland.blogs.com/EcmaScript.html#prod99) ( ";" )? |
| JScriptVarDeclarationList | ::= | [JScriptVarDeclaration](https://tomcopeland.blogs.com/EcmaScript.html#prod100) ( "," [JScriptVarDeclaration](https://tomcopeland.blogs.com/EcmaScript.html#prod100) )\* |
| JScriptVarDeclaration | ::= | [Identifier](https://tomcopeland.blogs.com/EcmaScript.html#prod4) ":" <IDENTIFIER\_NAME> ( [Initialiser](https://tomcopeland.blogs.com/EcmaScript.html#prod84) )? |
| insertSemiColon | ::= | *java code* |

**BNF de GO**

{

parserClass='com.goide.parser.GoParser'

classHeader='copyrightHeader.java'

implements='com.goide.psi.GoCompositeElement'

extends='com.goide.psi.impl.GoCompositeElementImpl'

elementTypeHolderClass='com.goide.GoTypes'

elementTypeClass='com.goide.psi.GoCompositeElementType'

tokenTypeClass='com.goide.psi.GoTokenType'

psiTreeUtilClass='com.goide.psi.GoPsiTreeUtil'

psiClassPrefix='Go'

psiImplClassSuffix='Impl'

psiPackage='com.goide.psi'

psiImplPackage='com.goide.psi.impl'

psiImplUtilClass="com.goide.psi.impl.GoPsiImplUtil"

parserUtilClass="com.goide.parser.GoParserUtil"

tokens=[

LBRACE = '{'

RBRACE = '}'

LBRACK = '['

RBRACK = ']'

LPAREN = '('

RPAREN = ')'

COLON = ':'

SEMICOLON = ';'

COMMA = ','

EQ = '=='

ASSIGN = '='

NOT\_EQ = '!='

NOT = '!'

PLUS\_PLUS = '++'

PLUS\_ASSIGN = '+='

PLUS = '+'

MINUS\_MINUS = '--'

MINUS\_ASSIGN = '-='

MINUS = '-'

COND\_OR = '||'

BIT\_OR\_ASSIGN = '|='

BIT\_CLEAR\_ASSIGN = '&^='

BIT\_CLEAR = '&^'

COND\_AND = '&&'

BIT\_AND\_ASSIGN = '&='

BIT\_AND = '&'

BIT\_OR = '|'

SHIFT\_LEFT\_ASSIGN = '<<='

SHIFT\_LEFT = '<<'

SEND\_CHANNEL = '<-'

LESS\_OR\_EQUAL = '<='

LESS = '<'

BIT\_XOR\_ASSIGN = '^='

BIT\_XOR = '^'

MUL\_ASSIGN = '\*='

MUL = '\*'

QUOTIENT\_ASSIGN = '/='

QUOTIENT = '/'

REMAINDER\_ASSIGN = '%='

REMAINDER = '%'

SHIFT\_RIGHT\_ASSIGN = '>>='

SHIFT\_RIGHT = '>>'

GREATER\_OR\_EQUAL = '>='

GREATER = '>'

VAR\_ASSIGN = ':='

TRIPLE\_DOT = '...'

DOT = '.'

SEMICOLON\_SYNTHETIC = '<NL>'

TYPE\_ = 'type'

raw\_string = 'raw\_string'

]

extends("(Or|And|Add|Conversion|Mul|Selector|Conditional)Expr")=BinaryExpr

extends(".\*(Expr|Literal|Lit|OperandName)")=Expression

implements("(Type|Import)Spec|(Var|Const|Param|(Anonymous)?Field|Label)Definition|Receiver")="com.goide.psi.GoNamedElement"

implements("FunctionLit")="com.goide.psi.GoSignatureOwner"

implements("MethodSpec")="com.goide.psi.GoNamedSignatureOwner"

implements("(Function|Method)Declaration")="com.goide.psi.GoFunctionOrMethodDeclaration"

implements("(Const|Type|Var)Declaration")="com.goide.psi.GoTopLevelDeclaration"

implements(".\*ReferenceExpression|FieldName")="com.goide.psi.GoReferenceExpressionBase"

extends("(Function|Method)Declaration")="com.goide.psi.impl.GoFunctionOrMethodDeclarationImpl<?>"

extends("(Type|Import|Method)Spec|(Var|Const|Param|(Anonymous)?Field|Label)Definition|Receiver")="com.goide.psi.impl.GoNamedElementImpl<?>"

extends("PackageClause|Parameters|Signature|ParameterDeclaration|Result|Type|(Const|Var|Import)Spec")="com.goide.psi.impl.GoStubbedElementImpl<?>"

elementTypeFactory("PackageClause|(Function|Method|Parameter)Declaration|(Type|Import|Method|Const|Var)Spec|RangeClause|RecvStatement|ShortVarDeclaration|(Var|Const|Param|(Anonymous)?Field|Label)Definition|Receiver|Parameters|Signature|Result|.\*Type|TypeList.\*")='com.goide.stubs.GoElementTypeFactory.stubFactory'

pin(".\*Statement")=1

extends(".\*Statement")=Statement

extends(".\*Type|TypeList")=Type

pin('.\*List(?:\_\d.\*)?')=1

generateTokenAccessors=true

}

| File | ::= | PackageClause semi ImportList TopLevelDeclaration\* {pin(".\*")=1} |
| --- | --- | --- |
| ImportList | ::= | (ImportDeclaration semi)+|<<emptyImportList>> {  methods=[addImport]  } |
| private semi | ::= | '<NL>' | ';' | <<eof>> |
| PackageClause | ::= | package identifier {pin=1 methods=[getName] stubClass="com.goide.stubs.GoPackageClauseStub"} |
| ImportDeclaration | ::= | import ( ImportSpec | '(' ImportSpecs? ')' ) { pin(".\*")=1 methods=[addImportSpec] } |
| private ImportSpecs | ::= | ImportSpec (semi ImportSpec)\* semi? {pin=1} |
| ImportSpec | ::= | [ '.' | identifier ] ImportString {  stubClass="com.goide.stubs.GoImportSpecStub"  methods=[getAlias getLocalPackageName shouldGoDeeper isForSideEffects isDot getPath getName isCImport]  } |
| ImportString | ::= | StringLiteral {methods=[getReferences resolve getPath getPathTextRange]} |
| Type | ::= | TypeName | TypeLit | ParType {stubClass="com.goide.stubs.GoTypeStub" methods=[getUnderlyingType shouldGoDeeper]} |
| ParType | ::= | '(' Type ')' {methods=[getActualType]} |
| private TypeName | ::= | TypeReferenceExpression QualifiedTypeReferenceExpression? |
| TypeReferenceExpression | ::= | identifier { methods=[getReference getQualifier resolve resolveType] } |
| left QualifiedTypeReferenceExpression | ::= | '.' identifier {elementType=TypeReferenceExpression} |
| private TypeLit | ::= | ArrayOrSliceType  | StructType  | PointerType  | FunctionType  | InterfaceType  | MapType  | ChannelType |
| ArrayOrSliceType | ::= | '[' ('...'|Expression?) ']' Type {pin=1} |
| StructType | ::= | struct '{' Fields? '}' {pin=1} |
| private Fields | ::= | FieldDeclaration (semi FieldDeclaration)\* semi? {pin=1} |
| FieldDeclaration | ::= | (FieldDefinitionList Type | AnonymousFieldDefinition) Tag? |
| Tag | ::= | StringLiteral |
| private FieldDefinitionList | ::= | FieldDefinition (',' FieldDefinition)\* |
| FieldDefinition | ::= | identifier { stubClass="com.goide.stubs.GoFieldDefinitionStub"} |
| AnonymousFieldDefinition | ::= | Type {  stubClass="com.goide.stubs.GoAnonymousFieldDefinitionStub"  methods=[getIdentifier getName getTypeReferenceExpression getGoTypeInner] } |

| PointerType | ::= | '\*' Type {pin=1} |
| --- | --- | --- |
| FunctionType | ::= | func Signature {pin=1 implements="com.goide.psi.GoSignatureOwner"} |
| Signature | ::= | Parameters Result? {  pin=1  stubClass="com.goide.stubs.GoSignatureStub"} |
| Result | ::= | '(' TypeListNoPin ')' | Type | Parameters {  // todo: reorder and pin (  stubClass="com.goide.stubs.GoResultStub"  methods=[isVoid]  } |
| Parameters | ::= | '(' [ (ParameterList ','?| TypeListNoPin) ] ')' {  pin=1  stubClass="com.goide.stubs.GoParametersStub"  } |
| private ParameterList | ::= | ParameterDeclaration (',' (ParameterDeclaration | &')'))\* {pin(".\*")=1} |
| ParameterDeclaration | ::= | ParamDefinitionListNoPin? '...'? Type | Type {  // todo: use left rule  stubClass="com.goide.stubs.GoParameterDeclarationStub"  methods = [isVariadic]  } |
| private ParamDefinitionListNoPin | ::= | ParamDefinition &(!('.' | ')')) (',' ParamDefinition)\* // todo |
| ParamDefinition | ::= | identifier {  stubClass="com.goide.stubs.GoParamDefinitionStub"  methods = [isVariadic]  } |
| InterfaceType | ::= | interface '{' MethodSpecs? '}' {  pin(".\*")=1  methods=[getMethods getBaseTypesReferences]  } |
| private MethodSpecs | ::= | MethodSpec (semi MethodSpec)\* semi? {pin=1} |
| MethodSpec | ::= | TypeName &(!'(') | identifier Signature {  stubClass="com.goide.stubs.GoMethodSpecStub"  methods=[getGoTypeInner getName]  } |
| MapType | ::= | map '[' Type ']' Type {  pin=1  methods = [  keyType="Type[0]"  valueType="Type[1]"  ]  } |
| ChannelType | ::= | ChanTypePrefix Type {pin=1} |
| private ChanTypePrefix | ::= | chan '<-'? | '<-' chan {pin(".\*")=1} |
| BlockWithConsume | ::= | <<consumeBlock>> | BlockInner {elementType=Block} |
| Block | ::= | BlockInner {methods=[processDeclarations]} |
| private BlockInner | ::= | '{' ('}' | (<<withOff Statements "BLOCK?" "PAR">> | (!() Statements)) '}') {pin(".\*")=1} |
| private Statements | ::= | StatementWithSemi\* |
| private StatementWithSemi | ::= | Statement (semi|&'}') { pin=1 recoverWhile=StatementRecover } |
| Statement | ::= | ConstDeclaration  | TypeDeclaration  | VarDeclaration  | LabeledStatement  | SimpleStatement  | GoStatement  | ReturnStatement  | BreakStatement  | ContinueStatement  | GotoStatement  | FallthroughStatement  | Block  | IfStatement  | SwitchStatement  | SelectStatement  | ForStatement  | DeferStatement {methods=[processDeclarations]} |
| private StatementRecover | ::= | !('!' | '&' | '(' | '\*' | '+' | '-' | ';' | '<-' | '^' | 'type' | '{' | '|' | '|=' | '||' | '}' | break | case | char | const | continue | decimali | default | defer | else | fallthrough | float | floati | for | func | go | goto | hex | identifier | if | int | interface | map | oct | return | select | string | raw\_string | struct | switch | var) |
| SimpleStatement | ::= | ShortVarDeclaration  | IncDecStatement  | (LeftHandExprList [AssignmentStatement | SendStatement]) {pin(".\*")=LeftHandExprList} |
| private TopLevelDeclaration | ::= | !<<eof>> TopDeclaration semi {pin=1 recoverWhile=TopLevelDeclarationRecover} |
| private TopDeclaration | ::= | ConstDeclaration  | TypeDeclaration  | VarDeclaration  | FunctionDeclaration  | MethodDeclaration  private TopLevelDeclarationRecover ::= !(';' |'type' | const | func | var) |
| ConstDeclaration | ::= | const ( ConstSpec | '(' ConstSpecs? ')' ) {pin(".\*")=1 methods=[addSpec deleteSpec]} |
| private ConstSpecs | ::= | ConstSpec (semi ConstSpec)\* semi? {pin=1} |
| ConstSpec | ::= | ConstDefinitionList [ ('=' ExpressionList | Type '=' ExpressionList) ] {  pin(".\*")=1  stubClass="com.goide.stubs.GoConstSpecStub"  methods=[deleteDefinition]  } |
| private ExpressionOrTypeWithRecover | ::= | ExpressionOrLiteralTypeExpr {recoverWhile=ExpressionListRecover} |
| private ExpressionOrTypeWithRecover2 | ::= | <<withOn "PAR" ExpressionOrTypeWithRecover>> | (!() ExpressionOrLiteralTypeExpr) {recoverWhile=ExpressionListRecover} |
| private ExpressionOrTypeWithRecover | ::= | ExpressionOrLiteralTypeExpr {recoverWhile=ExpressionListRecover} |
| private ExpressionOrLiteralTypeExpr | ::= | Expression | LiteralTypeExpr |
| private ExpressionList | ::= | ExpressionWithRecover (',' (ExpressionWithRecover | &')'))\* {pin(".\*")=1} |
| private ExpressionWithRecover | ::= | Expression {recoverWhile=ExpressionListRecover} |
| private ExpressionListRecover | ::= | !('!' | '!=' | '%' | '%=' | '&&' | '&' | '&=' | '&^' | '&^=' | '(' | ')' | '\*' | '\*=' | '+' | '++' | '+=' | ',' | '-' | '--' | '-=' | '...' | '/' | '/=' | ':' | ';' | '<' | '<-' | '<<' | '<<=' | '<=' | '<NL>' | '=' | '==' | '>' | '>=' | '>>' | '>>=' | '[' | ']' | '^' | '^=' | 'type' | '{' | '|' | '|=' | '||' | '}' | break | case | chan | char | const | continue | decimali | default | defer | else | fallthrough | float | floati | for | func | go | goto | hex | identifier | if | int | interface | map | oct | return | select | string | raw\_string | struct | switch | var) |
| TypeDeclaration | ::= | 'type' ( TypeSpec | '(' TypeSpecs? ')' ) {pin(".\*")=1} |
| private TypeSpecs | ::= | TypeSpec (semi TypeSpec)\* semi? {pin=1} |
| TypeSpec | ::= | SpecType {  methods=[getGoTypeInner getMethods shouldGoDeeper identifier="SpecType/identifier"]  stubClass="com.goide.stubs.GoTypeSpecStub"  } |
| SpecType | ::= | identifier Type {pin=1 stubClass="com.goide.stubs.GoTypeStub" extends="com.goide.psi.impl.GoTypeImpl" implements=Type} |
| VarDeclaration | ::= | var ( VarSpec | '(' VarSpecs? ')' ) {pin(".\*")=1 methods=[addSpec deleteSpec]} |
| private VarSpecs | ::= | VarSpec (semi VarSpec)\* semi? {pin=1} |
| VarSpec | ::= | VarDefinitionList ( Type [ '=' ExpressionList ] | '=' ExpressionList ) {  pin(".\*")=1  methods=[processDeclarations deleteDefinition getRightExpressionsList]  stubClass="com.goide.stubs.GoVarSpecStub"  } |
| private VarDefinitionList | ::= | VarDefinition ( ',' VarDefinition )\* |
| VarDefinition | ::= | identifier {  methods=[getGoTypeInner getReference getValue]  stubClass="com.goide.stubs.GoVarDefinitionStub"  } |
| ShortVarDeclaration | ::= | VarDefinitionList ':=' ExpressionList {pin=2 extends=VarSpec} |
| FunctionDeclaration | ::= | func identifier Signature BlockWithConsume? {  pin=2  stubClass="com.goide.stubs.GoFunctionDeclarationStub"  } |
| MethodDeclaration | ::= | func Receiver identifier Signature BlockWithConsume? {  pin=1  stubClass="com.goide.stubs.GoMethodDeclarationStub"  methods=[getReceiverType]  } |
| Receiver  } | ::= | '(' (identifier ReceiverTail | ReceiverTail) ')' {  methods=[getGoTypeInner]  pin=1  stubClass="com.goide.stubs.GoReceiverStub" |
| private ReceiverTail | ::= | Type ','? |
| Literal | ::= | int  | float  | floati  | decimali  | hex  | oct  | StringLiteral  | char |
| StringLiteral | ::= | string | raw\_string {  implements = "com.intellij.psi.PsiLanguageInjectionHost"  methods = [isValidHost updateText createLiteralTextEscaper getDecodedText]  } |
| OperandName | ::= | ReferenceExpression QualifiedReferenceExpression? {elementType=ReferenceExpression} |
| left QualifiedReferenceExpression | ::= | '.' identifier {elementType=ReferenceExpression} |
| ReferenceExpressiongetReadWriteAccess] extends=Expression} | ::= | identifier { methods=[getReference getQualifier resolve |
| CompositeLit | ::= | (LiteralTypeExprInner LiteralValue | TypeName LiteralValue) {pin(".\*")="LiteralTypeExprInner"} // the brackets are valuable |
| private LiteralTypeExprInner | ::= | StructType | ArrayOrSliceType | MapType |
| LiteralTypeExpr | ::= | LiteralTypeExprInner | TypeName |
| LiteralValue | ::= | (<<isModeOff "BLOCK?">> | <<isModeOn "PAR">> | <<prevIsType>>) '{' ElementList? '}' {pin=2} |
| private ElementList | ::= | E ( ',' E? )\* |
| private E | ::= | <<withOn "PAR" Element>> | (!() Element) {recoverWhile=E\_recover} |
| private E\_recover | ::= | !('}'|',') |
| Element | ::= | First [':' Value] {pin(".\*")=1} |
| private First | ::= | Key | <<keyOrValueExpression>> |
| Key | ::= | (FieldName &':') | !() Expression // nullability hack for key without reference inside |
| FieldName | ::= | identifier { methods=[getReference getQualifier resolve] } |
| Value | ::= | Expression | LiteralValue |
| FunctionLit | ::= | func Signature Block {pin=1 methods=[processDeclarations]} |
| Expression | ::= | OrExpr  | AndExpr  | ConditionalExpr  | AddExpr  | MulExpr  | UnaryExpr  | ConversionExpr  | MaxGroup  | ParenthesesExpr {methods=[getGoType] implements="com.goide.psi.GoTypeOwner"} |
| fake BinaryExpr | ::= | Expression+ {  methods=[left="Expression[0]" right="Expression[1]" getOperator]  } |
| OrExpr | ::= | Expression '||' Expression |
| AndExpr | ::= | Expression '&&' Expression |
| UnaryExpr | ::= | UnaryOp Expression {methods=[getOperator]} |
| ConditionalExpr | ::= | Expression RelOp Expression |
| AddExpr | ::= | Expression AddOp Expression |
| MulExpr | ::= | Expression MulOp Expression |
| ParenthesesExpr | ::= | '(' <<enterMode "PAR">> Expression <<exitModeSafe "PAR">>')' {pin=1} |
| BuiltinCallExpr | ::= | ReferenceExpression <<isBuiltin>> BuiltinArgumentList {pin=2} |
| BuiltinArgumentList | ::= | '(' [ BuiltinArgsInner ','? ] ')' {pin=1 extends=ArgumentList} |
| private BuiltinArgsInner | ::= | Type [ ',' BuiltinArgsTail ] | BuiltinArgsTail |
| private BuiltinArgsTail | ::= | ExpressionList '...'? |
| private MaxGroup | ::= | CompositeLit  | OperandName  | BuiltinCallExpr  | CallExpr  | TypeAssertionExpr  | SelectorExpr  | IndexOrSliceExpr  | Literal  | FunctionLit |
| private RelOp | ::= | '==' | '!=' | '<' | '<=' | '>' | '>=' |
| private AddOp | ::= | '+' | '-' | '|' | '^' |
| private MulOp | ::= | '\*' | '/' | '%' | '<<' | '>>' | '&' | '&^' |
| private UnaryOp | ::= | '+' | '-' | '!' | '^' | '\*' | '&' | '<-' |
| SelectorExpr | ::= | Expression '.' !('(' 'type') Expression |
| IndexOrSliceExpr | ::= | Expression '[' <<enterMode "PAR">> (SliceExprBody | IndexExprBody) <<exitModeSafe "PAR">> ']' {pin=2 methods = [getExpression getIndices]} |
| private IndexExprBody | ::= | Expression SliceExprBodyInner? |
| private inner SliceExprBodyInner | ::= | SliceExprBody |
| private SliceExprBody | ::= | &':' (':' Expression ':' Expression) | (':' Expression?) |
| TypeAssertionExpr | ::= | Expression '.' '(' &(!'type') Type ')' |
| CallExpr | ::= | Expression ArgumentList |
| ArgumentList | ::= | '(' [ ExpressionArgList '...'? ','? ] ')' {pin=1} |
| ConversionExpr | ::= | &ConversionPredicate Type ConversionTail |
| private ConversionPredicate | ::= | ConversionStart | '(' ConversionStart |
| private ConversionTail | ::= | '(' Expression ','? ')' {pin=1} |
| private ConversionStart | ::= | '\*' | '<-' | '[' | chan | func | interface | map | struct |
| LabeledStatement | ::= | LabelDefinition ':' Statement? {pin=2} |
| LabelDefinition | ::= | identifier {  stubClass="com.goide.stubs.GoLabelDefinitionStub" // todo: remove when you'll have some spare time  } |
| left SendStatement | ::= | '<-' Expression {pin=1 methods=[getSendExpression]} |
| left AssignmentStatement | ::= | assign\_op ExpressionList {pin=1} |
| IncDecStatement | ::= | Expression ('++' | '--') {pin=2} |
| assign\_op | ::= | '=' | '+=' | '-=' | '|=' | '^=' | '\*=' | '/=' | '%=' | '<<=' | '>>=' | '&=' | '&^=' |
| IfStatement | ::= | if Condition Block ElseStatement? |
| ElseStatement | ::= | else (IfStatement | Block) |
| SwitchStatement | ::= | SwitchStart (TypeSwitchStatement | ExprSwitchStatement) |
| SwitchStart | ::= | switch |
| left ExprSwitchStatement | ::= | Condition '{' ExprCaseClause\* '}' {pin=2 extends=SwitchStatement} |
| private Condition | ::= | <<enterMode "BLOCK?">> SimpleStatementOpt Expression? <<exitModeSafe "BLOCK?">> |
| private SimpleStatementOpt | ::= | [SimpleStatement ';'?] // todo: remove ? |
| ExprCaseClause | ::= | !'}' ExprSwitchCase ':' Statements? {pin=1 implements="com.goide.psi.GoCaseClause"} |
| private ExprSwitchCase | ::= | case ExpressionList | default {pin(".\*")=1} |
| left TypeSwitchStatement | ::= | (TypeSwitchGuard | SimpleStatement ';'? TypeSwitchGuard) '{' ( TypeCaseClause )\* '}' {pin=1 extends=SwitchStatement} |
| TypeSwitchGuard | ::= | [ VarDefinition ':=' ] Expression '.' TypeGuard |
| TypeGuard | ::= | '(' 'type' ')' {pin=2} |
| TypeCaseClause | ::= | !'}' TypeSwitchCase ':' Statements? {pin=1 implements="com.goide.psi.GoCaseClause"} |
| private TypeSwitchCase | ::= | case TypeList | default {pin(".\*")=1} |
| TypeList | ::= | Type ( ',' Type )\* ','? |
| TypeListNoPin | ::= | Type ( ',' Type )\* ','? {elementType=TypeList} |
| ForStatement | ::= | for <<enterMode "BLOCK?">> (ForOrRangeClause Block | Block | Expression Block) <<exitModeSafe "BLOCK?">> {pin(".\*")="for|ForOrRangeClause"} |
| private ForOrRangeClause | ::= | (ForClause | RangeClause) |
| ForClause | ::= | SimpleStatement? ';' Expression? ';' SimpleStatement? {methods=[processDeclarations]} |
| RangeClause | ::= | range Expression | ExpressionsOrVariables range Expression {pin(".\*")=1 extends=VarSpec methods=[getRangeExpression getLeftExpressionsList getRightExpressionsList]} |
| private ExpressionsOrVariables | ::= | ExpressionList '=' | VarDefinitionList ':=' |
| GoStatement | ::= | go Expression |
| SelectStatement | ::= | select '{' CommClause\* '}' |
| CommClause | ::= | CommCase ':' Statements? {pin=1} |
| CommCase | ::= | case ( Expression SendStatement | RecvStatement ) | default {pin(".\*")="case"} |
| RecvStatement | ::= | ExpressionsOrVariables? Expression {pin=0 extends=VarSpec methods=[getRecvExpression getLeftExpressionsList getRightExpressionsList]} |
| ReturnStatement | ::= | return ExpressionList? |
| LabelRef | ::= | identifier { methods=[getReference] } |
| BreakStatement | ::= | break LabelRef? |
| ContinueStatement | ::= | continue LabelRef? |
| GotoStatement | ::= | goto LabelRef |
| FallthroughStatement | ::= | fallthrough |
| DeferStatement | ::= | defer Expression |