

perl

Rules

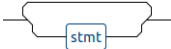
- compound_stmt
- stmt
- assignment_stmt
- if_stmt
- while_stmt
- until_stmt
- expr
- variable
- mul_div_op
- add_sub_op
- rel_op
- IF
- ELSE
- ELSE_IF
- WHILE
- UNTIL
- number
- sign
- IDENTIFIER
- INTEGER
- REAL
- MUL_OP
- DIV_OP
- ADD_OP
- SUB_OP
- POW_OP
- MOD_OP
- ASSIGN
- EQ_OP
- NE_OP
- LT_OP
- LE_OP
- GT_OP
- GE_OP
- NEWLINE
- WS

compound_stmt [Top](#)

Text notation:

compound_stmt : (stmt)+ ;

Visual notation:

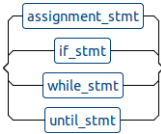


stmt [Top](#)

Text notation:

stmt : assignment_stmt | if_stmt | while_stmt | until_stmt ;

Visual notation:



assignment_stmt [Top](#)

Text notation:

assignment_stmt : variable ASSIGN expr ';' ;

Visual notation:



if_stmt [Top](#)

Text notation:

if_stmt : IF '(' expr ')' '{' stmt '}' (ELSE_IF '(' expr ')' '{' stmt '}')+ (ELSE '{' stmt '}')? ;

Visual notation:



while_stmt [Top](#)

Text notation:

while_stmt : WHILE '(' expr ')' '{' stmt '}' ;

Visual notation:

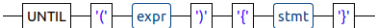


until_stmt [Top](#)

Text notation:

until_stmt : UNTIL '(' expr ')' '{' stmt '}' ;

Visual notation:

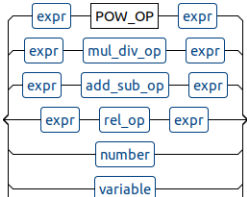


expr [Top](#)

Text notation:

expr : expr POW_OP expr | expr mul_div_op expr | expr add_sub_op expr | expr rel_op expr | number | variable | '(' expr ')' ;

Visual notation:



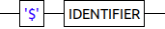


variableTop

Text notation:

variable : '\$' IDENTIFIER ;

Visual notation:

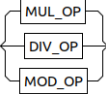


mul_div_opTop

Text notation:

mul_div_op : MUL_OP | DIV_OP | MOD_OP ;

Visual notation:

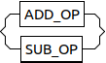


add_sub_opTop

Text notation:

add_sub_op : ADD_OP | SUB_OP ;

Visual notation:

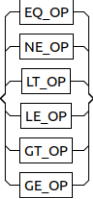


rel_opTop

Text notation:

rel_op : EQ_OP | NE_OP | LT_OP | LE_OP | GT_OP | GE_OP ;

Visual notation:



IFTop

Text notation:

IF : 'if' ;

Visual notation:



ELSETop

Text notation:

ELSE : 'else' ;

Visual notation:



ELSE_IFTop

Text notation:

ELSE_IF : 'elsif' ;

Visual notation:



WHILETop

Text notation:

```
WHILE : 'while' ;
```

Visual notation:

**UNTIL** [Top](#)

Text notation:

```
UNTIL : 'until' ;
```

Visual notation:

**number** [Top](#)

Text notation:

```
number : sign? (INTEGER | REAL) ;
```

Visual notation:

**sign** [Top](#)

Text notation:

```
sign : '+' | '-' ;
```

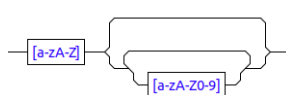
Visual notation:

**IDENTIFIER** [Top](#)

Text notation:

```
IDENTIFIER : [a-zA-Z][a-zA-Z0-9]* ;
```

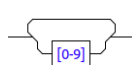
Visual notation:

**INTEGER** [Top](#)

Text notation:

```
INTEGER : [0-9]+ ;
```

Visual notation:

**REAL** [Top](#)

Text notation:

```
REAL : INTEGER '.' INTEGER ;
```

Visual notation:

**MUL_OP** [Top](#)

Text notation:

```
MUL_OP : '*' ;
```


Visual notation:

**DIV_OP** [Top](#)

Text notation:

DIV_OP : '/' ;

Visual notation:




ADD_OP

Top

Text notation:

ADD_OP : '+' ;

Visual notation:




SUB_OP

Top

Text notation:

SUB_OP : '-' ;

Visual notation:



POW_OP

Top

Text notation:

POW_OP : '**' ;

Visual notation:




MOD_OP

Top

Text notation:

MOD_OP : '%' ;

Visual notation:




ASSIGN

Top

Text notation:

ASSIGN : '=' ;

Visual notation:




EQ_OP

Top

Text notation:

EQ_OP : '==' ;

Visual notation:




NE_OP

Top

Text notation:

NE_OP : '!=' ;

Visual notation:




LT_OP

Top

Text notation:

LT_OP : '<' ;

Visual notation:




LE_OP

Top

Text notation:

LE_OP : '<=' ;

Visual notation:




GT_OP

Top

Text notation:

GT_OP : '>' ;

Visual notation:




GE_OP

Top

Text notation:

GE_OP : '>=' ;

Visual notation:




NEWLINE

Top

Text notation:

NEWLINE : '\\r'? '\\n' -> skip ;

Visual notation:



WS

Top

Text notation:

WS : ['\\t']+ -> skip ;

Visual notation:

