PythonsubsetParser.java

```
// Generated from c:/Users/Jorge
Melo/Desktop/python2asm_ProgSistBase1/grammar/PythonSubset.g4 by ANTLR 4.13.1
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.misc.*;
import org.antlr.v4.runtime.tree.*;
import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast",
"CheckReturnValue"})
public class PythonSubsetParser extends Parser {
 static { RuntimeMetaData.checkVersion("4.13.1", RuntimeMetaData.VERSION); }

 protected static final DFA[] _decisionToDFA;
 protected static final PredictionContextCache _sharedContextCache =
  new PredictionContextCache();
 public static final int
  T__0=1, T__1=2, T__2=3, T__3=4, T__4=5, T__5=6, T__6=7, T__7=8, T__8=9,
  T__9=10, T__10=11, T__11=12, T__12=13, T__13=14, T__14=15, T__15=16, T__16=17,
  T__17=18, FOR=19, WHILE=20, IF=21, ELIF=22, ELSE=23, IN=24, AND=25, OR=26,
  NOT=27, TRUE=28, FALSE=29, INDENT=30, DEDENT=31, IDENTIFIER=32, INT=33,
  STRING=34, NEWLINE=35, WS=36, COMMENT=37;
 public static final int
  RULE_prog = 0, RULE_stmt = 1, RULE_simple_stmt = 2, RULE_compound_stmt = 3,
  RULE_assign_stmt = 4, RULE_expr_stmt = 5, RULE_for_stmt = 6, RULE_while_stmt =
7,
  RULE_if_stmt = 8, RULE_elif_clause = 9, RULE_else_clause = 10, RULE_iterable =
11,
  RULE_range_call = 12, RULE_range_args = 13, RULE_expr = 14, RULE_comparison =
15,
  RULE_arith_expr = 16, RULE_unary_expr = 17, RULE_power_expr = 18, RULE_atom_expr
= 19,
  RULE_arg_list = 20;
 private static String[] makeRuleNames() {
  return new String[] {
   "prog", "stmt", "simple_stmt", "compound_stmt", "assign_stmt", "expr_stmt",
   "for_stmt", "while_stmt", "if_stmt", "elif_clause", "else_clause", "iterable",
   "range_call", "range_args", "expr", "comparison", "arith_expr", "unary_expr",
   "power_expr", "atom_expr", "arg_list"
  };
 }
 public static final String[] ruleNames = makeRuleNames();

 private static String[] makeLiteralNames() {
  return new String[] {
   null, "'='", "':'", "'range'", "'('", "')'", "','", "'=='", "'!='", "'>='",
```

```
    "'<='", "'>'", "'<'", "'+'", "'-'", "'*'", "'/'", "'%'", "'**'", "'for'",
    "'while'", "'if'", "'elif'", "'else'", "'in'", "'and'", "'or'", "'not'",
    "'True'", "'False'", "'INDENT'", "'DEDENT'"
   };
  }
  private static final String[] _LITERAL_NAMES = makeLiteralNames();
  private static String[] makeSymbolicNames() {
   return new String[] {
    null, null, null, null, null, null, null, null, null, null, null, null,
    null, null, null, null, null, null, null, "FOR", "WHILE", "IF", "ELIF",
    "ELSE", "IN", "AND", "OR", "NOT", "TRUE", "FALSE", "INDENT", "DEDENT",
    "IDENTIFIER", "INT", "STRING", "NEWLINE", "WS", "COMMENT"
   };
  }
  private static final String[] _SYMBOLIC_NAMES = makeSymbolicNames();
  public static final Vocabulary VOCABULARY = new VocabularyImpl(_LITERAL_NAMES,
_SYMBOLIC_NAMES);

  /**
   * @deprecated Use {@link #VOCABULARY} instead.
   */
  @Deprecated
  public static final String[] tokenNames;
  static {
   tokenNames = new String[_SYMBOLIC_NAMES.length];
   for (int i = 0; i < tokenNames.length; i++) {
    tokenNames[i] = VOCABULARY.getLiteralName(i);
    if (tokenNames[i] == null) {
     tokenNames[i] = VOCABULARY.getSymbolicName(i);
    }

    if (tokenNames[i] == null) {
     tokenNames[i] = "<INVALID>";
    }
   }
  }

  @Override
  @Deprecated
  public String[] getTokenNames() {
   return tokenNames;
  }

  @Override

  public Vocabulary getVocabulary() {
   return VOCABULARY;
  }

  @Override
  public String getGrammarFileName() { return "PythonSubset.g4"; }

  @Override
  public String[] getRuleNames() { return ruleNames; }
```

```
@Override
public String getSerializedATN() { return _serializedATN; }

@Override
public ATN getATN() { return _ATN; }

public PythonSubsetParser(TokenStream input) {
 super(input);
 _interp = new ParserATNSimulator(this,_ATN,_decisionToDFA,_sharedContextCache);
}

@SuppressWarnings("CheckReturnValue")
public static class ProgContext extends ParserRuleContext {
 public TerminalNode EOF() { return getToken(PythonSubsetParser.EOF, 0); }
 public List<StmtContext> stmt() {
  return getRuleContexts(StmtContext.class);
 }
 public StmtContext stmt(int i) {
  return getRuleContext(StmtContext.class,i);
 }
 public ProgContext(ParserRuleContext parent, int invokingState) {
  super(parent, invokingState);
 }
 @Override public int getRuleIndex() { return RULE_prog; }
}

public final ProgContext prog() throws RecognitionException {
 ProgContext _localctx = new ProgContext(_ctx, getState());
 enterRule(_localctx, 0, RULE_prog);
 int _la;
 try {
  enterOuterAlt(_localctx, 1);
  {
  setState(43);
  _errHandler.sync(this);
  _la = _input.LA(1);
  do {
   {
   {
   setState(42);
   stmt();
   }
   }
  setState(45);
  _errHandler.sync(this);
  _la = _input.LA(1);
  } while ( (((_la) & ~0x3f) == 0 && ((1L << _la) & 65367728144L) != 0) );
  setState(47);
  match(EOF);
  }
 }
 catch (RecognitionException re) {
  _localctx.exception = re;
```

```
     _errHandler.reportError(this, re);
     _errHandler.recover(this, re);
    }
    finally {
     exitRule();
    }
    return _localctx;
   }
  }
```