**Домашняя работа 3**

**Задание 1:**

Реализовать синтаксический разбор придуманного вами конфигурационного языка, основанного на S-выражениях (в которых у нас есть только скобки, пробелы и значения различных типов) для представления информации о студентах и группах (см. задачу 1 из практики 3).

Формат должен быть основан на грамматике, приведенной ниже. Детали формата на основе S-выражений необходимо продумать самостоятельно и обязательно описать их в БНФ. Формат должен поддерживать комментарии, типы данных: число, имя, строка.

Разбор реализовать средствами типа Lex/Yacc на выбранном ЯП (SLY в Python, Flex/Bison в C/C++, ANTLR в Java, ...).

После разбора результат преобразовать в JSON-представление. Пример из ПР 3.1 у вас должен получиться на выходе таким же и это надо показать в решении.

Программа должна иметь интерфейс командной строки: входным аргументом является имя конфигурационного файла, а результатом выполнения является вывод на экран (stdout) результата трансляции в JSON.

Дополнительный плюс за поддержку принципа DRY (то есть в язык можно добавить переменные, функции и так далее).

Базовая грамматика, которую необходимо реализовать и расширить для своих нужд (комментарии, представление data...):

|  |
| --- |
| from sly import Lexer, Parser  import json  import codecs  import sys  class SLexer(Lexer):  tokens = {NUMBER, STRING, LPAR, RPAR, NAME, LFIG, RFIG, FOR, PATTERN}  ignore = '\t\r '  ignore\_newline = r'\n+'  ignore\_comment = r'\#.\*'  STRING = r'\"[\w\dа-яА-Я\_\-\.\,\s]+\"'  LPAR = r'\('  RPAR = r'\)'  LFIG = r'\{'  RFIG = r'\}'  NAME = r'[a-zA-Z\_][a-zA-Z0-9\_-]+'  NUMBER = r'\d+'  NAME["for"] = FOR  PATTERN = r'\"[\w\dа-яА-Я\_\-\.\,\s]+%[\w\dа-яА-Я\_\-\.\,\s]\*\"'  @\_(r'\d+')  def NUMBER(self, t):  t.value = int(t.value)  return t  class SParser(Parser):  tokens = SLexer.tokens  def init(self):  self.names = {}  @\_('object')  def program(self, p):  return p.object  @\_('LPAR s\_exp\_list RPAR')  def object(self, p):  d = {}  d.update(p.s\_exp\_list)  return d  @\_('sexp s\_exp\_list')  def s\_exp\_list(self, p):  d = {}  d.update(p.sexp)  d.update(p.s\_exp\_list)  return d  @\_('')  def s\_exp\_list(self, p):  return {}  @\_('NAME LPAR list\_value RPAR')  def sexp(self, p):  d = {}  d[p.NAME] = p.list\_value  return d  @\_('value list\_value')  def list\_value(self, p):  if (isinstance(p.list\_value, list)):  if (len(p.list\_value) != 0):  return [p.value] + p.list\_value  return p.value  return [p.value] + [p.list\_value]  @\_('')  def list\_value(self, p):  return []  @\_('NUMBER')  def value(self, p):  return p.NUMBER  @\_('STRING')  def value(self, p):  s = p.STRING  s = s[1:-1]  return s  @\_('object')  def value(self, p):  return p.object  @\_('LFIG PATTERN FOR NUMBER NUMBER RFIG')  def value(self, p):  b = p.NUMBER0  e = p.NUMBER1  word = p.PATTERN  word = word[1:-1]  parts = word.split('%')  a = []  for x in range(b, e+1):  a.append(parts[0] + str(x) + parts[1])  return a  def parse\_config\_file(input\_file, output\_file):  lexer = SLexer()  parser = SParser()  with open(input\_file, 'r', encoding='utf-8') as file:  config\_data = file.read()  result = parser.parse(lexer.tokenize(config\_data))  with open(output\_file, 'w', encoding='utf-8') as json\_file:  json.dump(result, json\_file, indent=2, ensure\_ascii=False)  if \_\_name\_\_ == '\_\_main\_\_':  import sys  if len(sys.argv) != 3:  print("Использование: python main.py <входной файл> <выходной файл>")  sys.exit(1)  input\_file = sys.argv[1]  output\_file = sys.argv[2]  parse\_config\_file(input\_file, output\_file) |

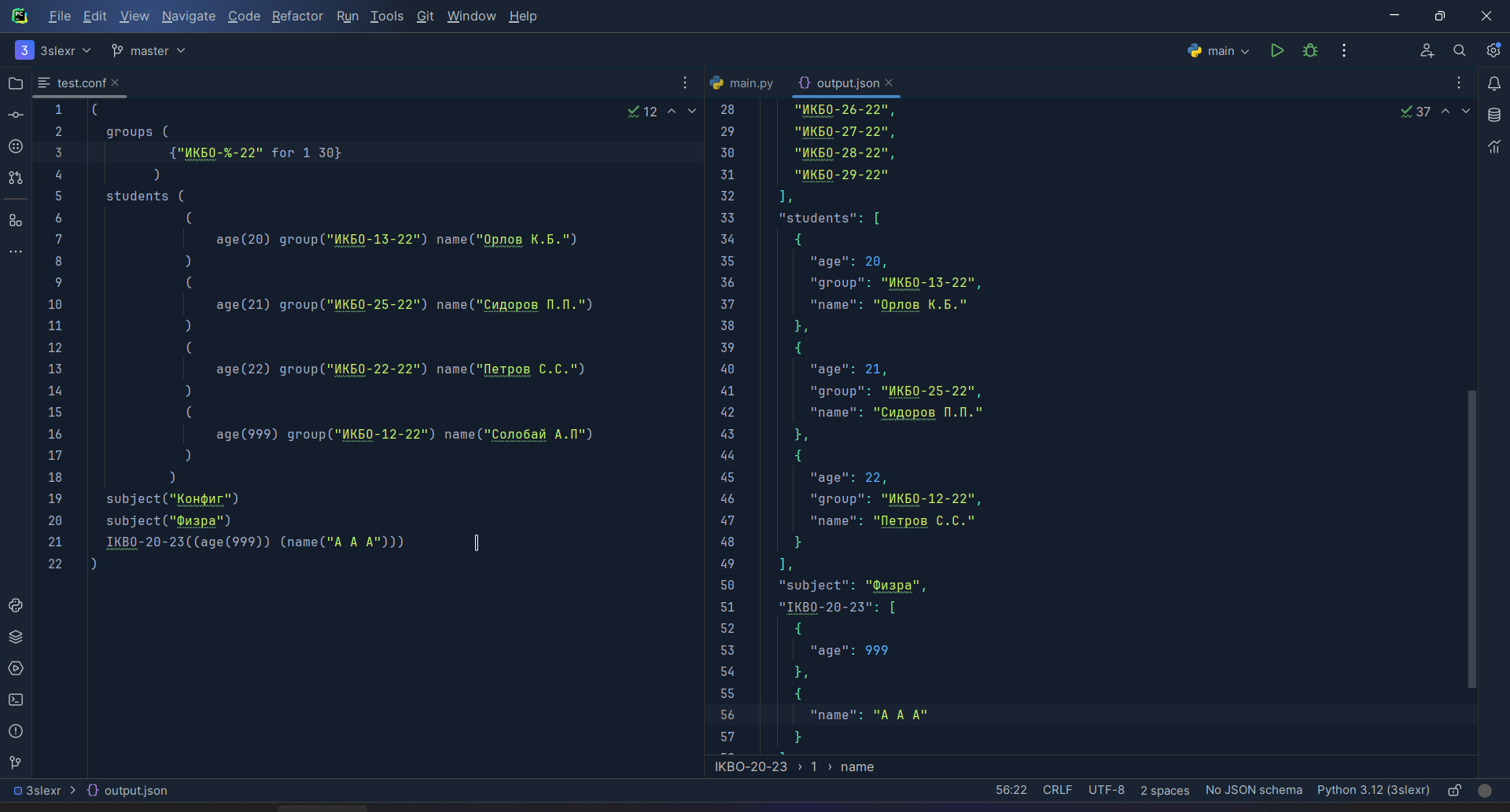


Рисунок 1 - Пример работы программы