**Содержание**

[**Введение** 2](#_Toc29744866)

[**Цель работы** 3](#_Toc29744867)

[**Логическое программирование** 4](#_Toc29744868)

[**PyDatalog** 5](#_Toc29744869)

[**pymorphy2** 6](#_Toc29744870)

[**Другие модули** 7](#_Toc29744871)

[**Заключение** 8](#_Toc29744872)

[**Приложение А. Листинг программы** 9](#_Toc29744873)

# **Введение**

**В настоящее время, в эпоху цифровых технологий, крайне стремительно развиваются различного рода информационные системы. Вырастает их количество, а также усложняется их составляющая.**

**На данный момент, одной достаточно популярной функциональностью системы является описание той или иной предметной области, в рамках которой в дальнейшем будут производиться различные операции, например, машинного обучения или связанные с искусственным интеллектом. Это приводит нас к задаче описания предметной области, которая является актуальной на данный момент.**

# **Цель работы**

**Целью данной курсовой работы является получение навыков описания предметной области методами и средствами PyDatalog.**

# **Логическое программирование**

Логическое программирование — парадигма программирования, основанная на автоматическом доказательстве теорем, с использованием механизмов логического вывода информации на основе заданных фактов и правил вывода. Язык Пролог и логическое программирование широко используются для создания баз знаний и экспертных систем и исследований в сфере искусственного интеллекта на основе логических моделей баз знаний и логических процедур вывода и принятия решений.

Самым известным языком логического программирования является Prolog.

Первым языком логического программирования был язык Planner, в котором была заложена возможность автоматического вывода результата из данных и заданных правил перебора вариантов (совокупность которых называлась планом). Planner использовался для того, чтобы понизить требования к вычислительным ресурсам (с помощью бэктрекинга — поиска с возвратом) и обеспечить возможность вывода фактов, без активного использования стека. Затем был разработан язык Prolog, который не требовал плана перебора вариантов и был, в этом смысле, упрощением языка Planner.

От языка Planner также произошли логические языки программирования QA-4, Popler, Conniver и QLISP. Языки программирования Mercury, Visual Prolog, Oz и Fril произошли уже от языка Prolog. На базе языка Planner было разработано также несколько альтернативных языков логического программирования, не основанных на методе поиска с возвратами, например, Ether.

# **PyDatalog**

**PyDatalog** – модуль в языке Python, интегрирующий Datalog, унаследованный от Prolog.

Использование pyDatalog позволяет описать предметную область на основе логической парадигмы. Для этого есть некоторые инструменты, например, факты, правила, термы.

Импортировать модуль можно следующей командой

*from* pyDatalog *import* pyDatalog *as* pyDt

**Термы** – подобие переменных, которые будут использовать в фактах и правилах. Объявить их можно при помощи специальной команды

pyDt.create\_terms()

**Факты** – это некая информация об объектах(термах), ее можно «записать» с помощью оператора «+»

+ student("артур")

+ student("василий")

**Правила** – основная логическая составляющая, позволяющая применить некие ограничения на вывод информации. Обычно в правилах присутствует следующий оператор «<=»

(attend[Y1] == X1) <= (preferedMuseum[Y1] == X1)

(lookat[Z1] == X1) <= (preferedArtist[Z1] == X1)

# **pymorphy2**

Также для выполнения курсовой работы был использован модуль pymorphy2.

**pymorphy2** - морфологический анализатор, разработанный на языке программирования Python. Выполняет лемматизацию и анализ слов, способен осуществлять склонение по заданным грамматическим характеристикам слов. Работает со словарём OpenCorpora, а для незнакомых слов строит гипотезы. Поддерживаются русский и украинский языки.

В курсовой работе данный модуль позволял определять во входящих вопросах, что нужно сделать, с чем нужно это сделать и тип задачи.

Для выполнения этих процедур необходимо было создать паттерны, которые анализатор искал бы в тексте. Паттерны можно прописать самому, ведь логику распознавания тоже придется написать нам.

 NOUN

 -a-

Здесь мы говорим, что ищем существительное и вводим его в качестве переменной.

# **Другие модули**

Также для выполнения курсовой работы мною были импортированы следующие модули:

* os; использовался для открытия и чтения файла с тестом;
* enums; использовался для определения перечислений с типами задачи, объектов задач и видов изменения;
* re; использовался для регулярных выражений и поиска по ним;
* json; для парсинга строки в dict.

**Заключение**

Итоговым результатом выполнения курсовой работы стала полностью работоспособная программа, способная различать из вопроса на входе объект преобразований, тип преобразований и вид задачи (прямую/обратную).

Программа была успешно протестирована – описанная предметная область теста и реализованные правила позволяют отвечать на вопросы из теста.

В ходе выполнения данной работы мною были получены навыки описания предметной области методами pyDatalog, также был получен опыт использования морфологического анализатора pymorphy2, работы с os и json.

**Приложение А. Листинг программы**

*import* os

*import* json

*from* pyDatalog *import* pyDatalog *as* pyDt

*from* collections *import* OrderedDict

*from* enums *import* Type

*import* re

*import* pprint

*import* pymorphy2 *as* py

*import* requests

*# 1. Считать из вопроса задание конкретное - что нужно сделать?  :  Запиши цифрами число ↓ Что сделать? В каком виде? С чем?/Что?*

*# 2. Спарсить данные из вопроса                                  :  четыреста шесть тысяч семьсот двенадцать ↓*

*# 3. Конвертировать их в ответ                                   :  406712 ↓*

*# Вопрос: "Запиши цифрами число четыреста шесть тысяч семьсот двенадцать"*

*# PyDatalog*

pyDt.create\_terms('X,Y,Z,name,X1,X2,X3,X4,museum,Y1,Y2,Y3,Y4,artist,Z1,Z2,Z3,Z4,prefer,attend,student,lookat,preferedMuseum,preferedArtist,students')

*class* Student(pyDt.Mixin):

*def* \_\_init\_\_(self, name=""):

        super(Student, *self*).\_\_init\_\_()

*self*.name = name

*def* \_\_repr\_\_(self):

*return* *f*'{*self*.name}'

*class* Museum(pyDt.Mixin):

*def* \_\_init\_\_(self, name=""):

        super(Museum, *self*).\_\_init\_\_()

*self*.name = name

*def* \_\_repr\_\_(self):

*return* *f*'{*self*.name}'

*class* Artist(pyDt.Mixin):

*def* \_\_init\_\_(self, name=""):

        super(Artist, *self*).\_\_init\_\_()

*self*.name = name

*def* \_\_repr\_\_(self):

*return* *f*'{*self*.name}'

+ student("артур")

+ student("василий")

+ museum('эрмитаж')

+ museum('бенуа')

+ artist('айвазовский')

+ artist('босх')

s0 = Student('василий')

s1 = Student('артур')

m0 = Museum('эрмитаж')

m1 = Museum('бенуа')

a0 = Artist('айвазовский')

a1 = Artist('босх')

students['артур'] = s1

students['василий'] = s0

preferedMuseum[m0] = s0

preferedMuseum[m1] = s0

preferedArtist[a0] = s0

preferedArtist[a1] = s1

*#rules*

(attend[Y1] == X1) <= (preferedMuseum[Y1] == X1)

(lookat[Z1] == X1) <= (preferedArtist[Z1] == X1)

*#-----------------------------------------------------------------------------------------*

*# Global variables::*

\_path = *f*'{os.path.dirname(os.path.abspath(\_\_file\_\_))}\\test.json'

\_pattern = re.compile(*r*'([А-ЯA-Z]((т.п.|т.д.|пр.|г.)|[^?!.\(]|\([^\)]\*\))\*[.?!])')

\_names = {}

\_result = []

\_morph = py.MorphAnalyzer()

\_token = 'trnsl.1.1.20191221T065111Z.4f507e2e1aa5093c.fcc75a438bbb14235e2a046d1a736c863ca65eeb'

\_url\_trans = 'https://translate.yandex.net/api/v1.5/tr.json/translate'

*# some patterns..*

\_source = '''

# Запиши цифрами число

# гл  сущ  сущ

# Что сделать? В каком виде? С чем/кем сделать?

 NOUN

 -a-

'''

*#+ fileReader(file, readFromJson())*

*#Read from Json::*

*def* readFromJson():

*with* open(\_path, 'r', encoding='utf-8') *as* \_file:

        \_data = json.load(\_file)['questions']

        parseQuestions(\_data)

*#PArse Questions::*

*def* parseQuestions(\_data):

*# search through questions and parse them*

    patterns = parseSource(\_source)

    \_text = ''

*for* q in \_data:

*# Question is q['question']*

*# Answers is q['answers']*

        question = q['question']

        inWhichType = []

        textResult = parseText(patterns, question)

*for* lineResult in textResult:

*for* questionResult in lineResult:

*# Здесь нужно обработать распознанные pymorphy данные*

                """

                r = {

                    'question': "",

                    'pattern' : ?,

                    'resultList' : ()

                }

                """

*#print(r['resultList'])*

*for* r in questionResult['resultList']:

*#print(type(r[1].tag))*

*#r[1] - элемент tuple(множества)*

*# r[1].tag.POS - часть речи*

                    """if r[1].tag.POS == 'VERB':

                        # Определение "Что сделать?"

                        for item in Convertation:

                            if r[1].normal\_form.upper() == item.value.upper(): # Сравниваем результаты аналииза pymorphy с хранящимися типаами заданий

                                whatShouldDo = item"""

*if* r[1].tag.POS == 'NOUN':

*#if r[1].tag.case == 'nomn':*

*# Определение "В каком виде?"*

*for* item in Type:

*if* r[1].normal\_form.upper() == item.value.upper(): *# Сравниваем результаты аналииза pymorphy с хранящимися типаами заданий*

                                inWhichType = item

                        """if r[1].tag.case == 'nomn':

                            # Определение "С чем сделать?"

                            for item in Target:

                                if r[1].normal\_form.upper() == item.value.upper(): # Сравниваем результаты аналииза pymorphy с хранящимися типаами заданий

                                    whatIsTheTarget = item"""

*#print(q)*

*#print(inWhichType)*

*if* inWhichType == Type.Museum:

        museum(q)

*elif* inWhichType == Type.Artist:

        artist(q)

*def* museum(q):

    answer = []

    names = []

    data = student(X).data

*for* s in data:

        names.append(s[0].upper())

    \_name = ''

*for* w in q['question'].split():

*if* w.upper() in names:

*for* a in ((attend[Y] == (students[w.lower()] == X).data[0][0]).data):

                answer.append(a[0])

    \_result.append(*f*'''

Вопрос: {q['question']}

Ответ:  {answer}''')

*def* artist(q):

*#print(q)*

    answer = []

    names = []

    data = student(X).data

*for* s in data:

        names.append(s[0].upper())

    \_name = ''

*for* w in q['question'].split():

*if* w.upper() in names:

*for* a in ((lookat[Z] == (students[w.lower()] == X).data[0][0]).data):

                answer.append(a[0])

    \_result.append(*f*'''

Вопрос: {q['question']}

Ответ:  {answer}''')

*class* PPattern:

*def* \_\_init\_\_(self):

        super().\_\_init\_\_()

*self*.tags = []

*self*.rules = []

*self*.example = ''

*def* checkPhrase(self, words, used=set()):

*def* getNextWord(wordList):

*if* len(wordList) == 0:

*return* None

            index = wordList[0]

            wordList[0:1] = []

*return* index

*def* checkWord(tags, word, prevResult):

            variants = \_morph.parse(word)

*for* v in variants:

*if* set(tags) <= v.tag.grammemes \

                        and *self*.checkRules(prevResult + [(word, v)]):

*return* (word, v)

*return* None

        result = []

        wordList = list(set([x *for* x in range(0, len(words))]) - used)

        wordList.sort()

        wi = getNextWord(wordList)

        nextTag = 0

        usedP = set()

*while* wi is not None:

            w = words[wi]

            res = checkWord(*self*.tags[nextTag].split(','), w, result)

*if* res is not None:

                result.append(res)

                usedP.add(wi)

                nextTag = nextTag + 1

*if* nextTag >= len(*self*.tags):

*return* (result, usedP)

            wi = getNextWord(wordList)

*return* None

*def* checkRules(self, result):

*for* r in *self*.rules:

            indexes = r[0]

            func = r[1]

            l = r[2]

*if* max(indexes) < len(result):  *# У нас есть достаточно данных*

                args = [result[x][1] *for* x in indexes]

*if* not func(\*args):

*return* False

*return* True

*def* checkPropRule(self, getFunc, getArgs, srcFunc, srcArgs, \

                      op=*lambda* x, y: x == y):

        v1 = getFunc(getArgs)

        v2 = srcFunc(srcArgs)

*return* op(v1, v2)

*def* setProp(self, setFunc, setArgs, srcFunc, srcArgs):

        setFunc(setArgs, srcFunc(srcArgs))

*import* io

*import* ast

*def* parseSource(src):

*def* parseFunc(expr, names):

        m = ast.parse(expr)

*# Получим список уникальных задействованных имен*

        varList = list(set([x.id *for* x in ast.walk(m) *if* type(x) == ast.Name]))

*# Найдем их позиции в грамматике*

        indexes = [names.index(v) *for* v in varList]

        lam = 'lambda %s: %s' % (','.join(varList), expr)

*return* (indexes, eval(lam), lam)

*def* parseLine(s):

        nonlocal arr, last

        s = s.strip()

*if* s == '':

            last = None

*return*

*if* s[0] == '#':

*return*

*if* last is None:

            last = PPattern()

            arr.append(last)

*if* s[0] == ':':  *# имена*

            names[s[1:]] = last

*elif* s[0] == '-':  *# внутренние имена*

            s = [x.strip('-') *for* x in s[1:].strip().split()]

            last.names = s

*elif* s[0] == '=':  *# правила*

            expr = s[1:].strip()

            last.rules.append(parseFunc(expr, last.names))

*else*:

            last.tags = s.split()

    arr = []

    last = None

    buf = io.StringIO(src)

    s = buf.readline()

*while* s:

        parseLine(s)

        s = buf.readline()

*return* arr

*def* parseText(pats, text):

*def* parseLine(line):

        words = line.split()

        used = set()

        was = False

        result = []

*for* p in pats:

            usedP = set()

*while* True:

                res = p.checkPhrase(words, usedP)

*if* res:

                    (res, newP) = res

                    used = used.union(newP)

                    first = list(newP)[0]

                    usedP = set([x *for* x in range(first + 1)])

                    print('+', line, p.tags, [r[0] *for* r in res])

                    was = True

                    result.append({

                        'question': line,

                        'pattern': p,

                        'resultList': res

                    })

*else*:

*break*

*if* not was:

            print('-', line)

*return* result

    buf = io.StringIO(text)

    s = buf.readline()

    resultList = []

*while* s:

        s = s.strip()

*if* s != '':

            resultList.append(parseLine(s))

        s = buf.readline()

*return* resultList

*def* tags(word):

    morph = py.MorphAnalyzer()

*return* morph.parse(word)

*def* main():

    readFromJson()

*for* r in \_result:

        print(r)

*if* \_\_name\_\_ == '\_\_main\_\_':

    main()