МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курганский государственный университет»

Кафедра «Программное обеспечение автоматизированных систем»

Функциональное и логическое программирование

Методические указания и контрольные задания
к выполнению контрольной работы
по дисциплине «Функциональное и логическое программирование»
для студентов очной и заочной формы обучения направления подготовки
09.03.04 «Программная инженерия»

Курган 2017

Кафедра: «Программное обеспечение автоматизированных систем».

Дисциплина: «Функциональное и логическое программирование»

(направление 09.03.04 «Программная инженерия»).

Составила: канд. техн. наук, доц. Т. Р. Змызгова.

Утверждены на заседании кафедры «26» апреля 2017 г.

Рекомендованы методическим советом университета « 12 » декабря 2016 г.

ВВЕДЕНИЕ

Задачами дисциплины «Функциональное и логическое программирование» являются изучение общих концепций и методов современного декларативного программирования, в частности, таких его разновидностей, как функциональное и логическое программирование, позволяющих эффективно решать задачи, связанные с обработкой символьной информации, нетипизированных данных, построения систем поддержки принятия решения, искусственного интеллекта, а также экспертных систем.

Для выполнения контрольной работы по дисциплине студент должен обладать знаниями синтаксиса и семантики логического языка программирования Prolog и функциональных возможностей языка программирования Python, а также иметь представление о составе и функциях операционных систем.

Контрольная работа должна содержать:

- титульный лист;
- оглавление;
- постановку задачи, в которой помимо формулировки задания определяются методы решения и выбирается оптимальный вариант построения программы;
- описание решения задачи, где представлен алгоритм решения, на основе которого создается программный продукт и разработаны тестовые задания к программе;
- список используемой литературы;
- приложение с листингом программы;
- приложение программы на электронном носителе информации, где должен быть записан исходный файл, объектный файл, загрузочный модуль программы.

Номера вариантов определяются по последней цифре номера зачетной книжки. Перед выполнением контрольной работы необходимо внимательно изучить методические указания и рекомендуемую литературу в соответствии с рабочей программой дисциплины.

Задание 1. Функциональное и логическое программирование

Подготовить реферат на заданную тему. Требования к оформлению: объем работы до 10 страниц. Формат MS Word; стиль Times New Roman; шрифт 12-14; интервал – одинарный; поля – 2 см. Тема реферата – заглавными буквами (шрифт 14, выравнивание – по центру). Реферат распечатать и представить в электронном виде.

1.1 Варианты заданий

Перечень тем для подготовки реферата

1 Python: особенности параллельного программирования на основе модели многопоточности.

2 Компьютерная графика в программах на языке Prolog.

3 Классы и наследование Python. Способы их взаимодействия через интерфейсы и иерархию.

4 Разработка программы графической иллюстрации решения задачи о Ханойских башнях.

5 Основы функционального программирования на Python. Лямбда-функции.

6 Разработка прототипа экспертной системы по классификации объектов некоторой предметной области на языке Prolog.

7 Метаклассы и динамические атрибуты Python.

8 Интеграция возможностей Prolog в Python.

9 Сетевые приложения на Python.

10 Возможности логического вывода в Python-программах.

Задание 2. Python: элементы функционального программирования

Освоение языка Python позволяет быстро создавать как прототипы программных систем, так и сами программные системы, помогает в интеграции программного обеспечения для решения научных и производственных задач, что необходимо для профессиональной подготовки бакалавров по направлению «Программная инженерия».

В результате изучения дисциплины студенты на примере доступного языка программирования Python осваивают объектно-ориентированное и функциональное программирование, позволяющие быстро перейти к решению задач в соответствующих предметных областях.

Тематика контрольных заданий предполагает исследовательскую часть: постановку задачи, анализ подходов к ее решению, создание алгоритма и практическую часть по реализации этого алгоритма на языке Python. При этом формируется умение увидеть прикладной аспект в решении конкретной научной задачи, грамотно представить и интерпретировать результат. В случае необходимости студент должен проанализировать полученный результат, скорректировать математическую модель, лежащую в основе задачи и решить задачу правильно и грамотно с использованием разработанных алгоритмов.

Замечание. При написании скриптов использовать функциональные возможности Python.

2.1 Функции

2.1.1 Элементы функционального программирования в Python

Функциональное программирование – это отдельная парадигма программирования, которая подразумевает написание программ, в которых единственным действием является вызов функции, при этом функциональная программа представляет собой некоторое выражение, а выполнение программы означает вычисление значения этого выражения.

При сравнении функционального и императивного подхода к программированию можно отметить следующие свойства функциональных программ:

• функциональные программы не используют переменные в том смысле, в котором это слово употребляется в императивном программировании. В частности, в функциональных программах не используется оператор присваивания;

• в функциональных программах нет циклов;

• выполнение последовательности команд в функциональной программе бессмысленно, поскольку одна команда не может повлиять на выполнение следующей;

• функциональные программы используют функции более высоких порядков (функции над функциями): функции можно передавать в другие в качестве аргументов и возвращать в качестве результата. В общем случае рекомендуется проводить вычисления, результатом которого будет функция;

• вместо циклов функциональные программы широко используют рекурсивные функции.

Элементы функционального программирования есть и в Python. Функции в Python являются основными программными структурами, которые обеспечивают структурирование программы и многократное использование программного кода. В Python есть встроенные функции, которые содержатся в ядре языка, функции, распределенные по модулям, кроме того, можно создавать функции, определённые пользователем.

В языке Python можно создать четыре типа функций: глобальные функции, локальные функции, лямбда-функции и методы.

Глобальные объекты (включая функции) доступны из любой точки программного кода в том же модуле, которому принадлежит объект. Глобальные объекты доступны также и из других модулей.

Локальные функции (вложенные) – функции, которые объявляются внутри других функций. Эти функции видимы только внутри тех функций, где они были объявлены, поэтому они особенно удобны для создания небольших вспомогательных функций, которые нигде больше не используются.

Лямбда-функции – это выражения, которые могут создаваться непосредственно в месте их использования; они имеют множество ограничений по сравнению с обычными функциями.

Методы – это те же функции, которые ассоциированы с определенным типом данных и могут использоваться только в связке с этим типом данных.

Функцию в Python можно создать двумя способами: с помощью оператора ***def*** и ***lambda***-выражения. Первый способ позволяет использовать операторы. При втором – определение функции может быть только выражением.

Составная инструкция определения функции (глобальной или локальной) ***def*** в языке Python имеет следующий синтаксис:

def имя_функции(список_формальных_параметров):
последовательность_выражений

Первая строка определения обычно называется заголовком функции, обозначенным ключевым словом ***def*** (от англ. *define* – определить). Правила выбо-

ра имен функций полностью аналогичны правилам выбора имен переменных. Список параметров определяет набор значений, которые могут быть переданы функции в качестве исходных данных; перечисляются через запятую. Заголовок функции в Python завершается двоеточием. После него может следовать любое количество выражений (тело функции), но они должны быть записаны со смещением относительно начала строки. Тело функции образует программный код, который выполняется интерпретатором всякий раз, когда производится вызов функции.

Оператор ***return*** используется для возврата из функции, т. е. для прекращения её работы и выхода из неё. При этом можно также вернуть некоторое значение из функции. Если мы вообще не используем инструкцию ***return***, функция будет возвращать значение ***None***. При вызове функции основная программа приостанавливает свою работу, пока функция не завершит свою и не передаст ей управление. Функции, вычисляющие какое-либо значение, возвращают его с помощью инструкции ***return***.

Пример. Функция factorial(), которая принимает один параметр – число, и возвращает значение – факториал этого числа (рисунок 1).

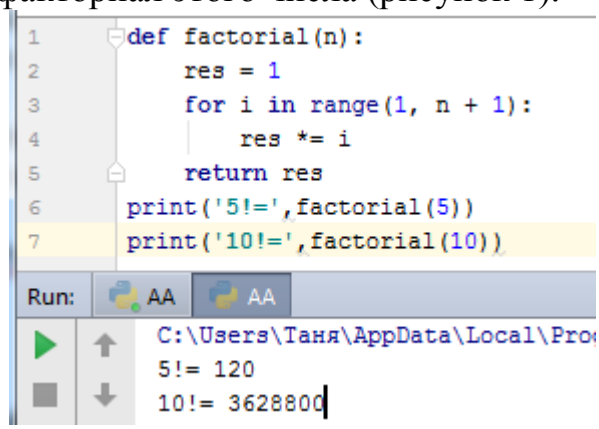

Рисунок 1 – Вычисление факториала с помощью функции ***def***

В отличие от других языков программирования при создании функции не указывается тип параметров. Если при объявлении функции какие-то параметры указаны в виде *имя_параметра=значение*, то это значение будет присвоено соответствующему параметру по умолчанию, если не будет присвоения при вызове. Параметры, не имеющие значений по умолчанию, должны быть указаны перед параметрами со значениями по умолчанию, иначе будет синтаксическая ошибка.

В языке Python существует возможность создавать ***функции без имени (анонимные)***, используя ключевое слово ***lambda***. Анонимная функция может содержать только одно выражение, которое должно всегда возвращать значение.

В отличие от обычной функции, созданной с помощью ключевого слова ***def***, ***lambda*** – функция возвращает объект, который разрешается присвоить переменной. Впоследствии она может быть использована для того, чтобы обратиться к функции (обратный вызов) в любом месте программы и исполнить блок выражений, которые содержит функция. Таким образом, конструкция

lambda позволяет программистам использовать альтернативный синтаксис для создания функции.

Lambda-функция в языке Python – это функция без имени, о которой известно только количество аргументов и формула для вычисления итогового значения, причем формула должна записываться единым выражением. Объекты функций в этом случае создаются в форме выражений. За ключевым словом ***lambda*** следуют один или более аргументов и далее, вслед за двоеточием, находится выражение:

lambda arg l, arg2, argN: выражение

Например, ***lambda*** – функция, складывающая три числа:

lambda x,y,z : x+y+z

Пример. Разницу между обычным определением функции и ***lambda***-функцией наиболее просто пояснить на примере работы программы, код которой приведен на рисунке 2.

```
1  def f1 (x):
2      return x**2
3  f2 = lambda x: x**2
4  print(f1(4 ), f2(4))
```

```
Run:  AA  AA  AA
C:\Users\Таня\AppData\Loc
16 16
```

Рисунок 2 – Вычисление с помощью ***lambd***а – функции

Такие конструкции удобны для создания малых функций и позволяют встраивать определения функций в программный код, который их использует. Аргумент после ключевого слова ***lambda*** стоит без скобок и выражение в данной функции не требует ключевого слова ***return***, поскольку ***lambda***–функция в любом случае возвращает значение.

Очевидно, что описывать огромную функцию, вызывающуюся много раз, ***lambda***-функцией, по меньшей мере, неразумно, но в некоторых случаях ***lambda***-функции бывают полезны. Например, в результате выполнения кода, приведенного на рисунке 3, на экран будет выдано 5.0.

```
1  R=lambda x,y : pow(x*x+y*y,0.5)
2  print(R(3,4))
```

Рисунок 3 – Применение ***lambda*** -функции

Ту же самую функцию можно объявить стандартным способом (рисунок 4):

```
1  def R(x,y):
2      return pow(x*x+y*y,0.5)
3  print(R(3,4))
```

Рисунок 4 – Применение функции ***def***

Применение ***lambda***-функций в контексте функционального программирования приводит к значительной экономии места и времени. Однострочные

lambda-функции часто используются для программирования кнопок в программах с графическим пользовательским интерфейсом.

Замечание 1. Поскольку ключевое слово ***lambda*** предлагает альтернативный способ для создания функции, он часто используется для встраивания функции в любое место кода. Например, обратные вызовы очень часто программируются в виде однострочных ***lambda***-выражений, встроенных непосредственно в список аргументов вместо обычного определения при помощи ***def*** и вызова по имени:

def function_1 : исполняемые-выражения
def function_2 : исполняемые-выражения
callbacks = [function_1 , function_2]

Такую запись можно выразить более кратко:

callbacks = [lambda : выражение , lambda : выражение]

Замечание 2. В отличие от обычной функции определение ***lambda***-функции может быть мгновенно сгенерировано программой:

```
def genincr(n):
    return lambda x, i=n: x+i
```

Эта функция возвратит ***функцию-инкрементатор***, увеличивающую свой аргумент на ***n***, где ***n*** задаётся при создании функции. Возможен и иной вариант:

```
genincr=lambda n:lambda x,i=n:x+i
```

Это определение функции ***genincr*** полностью эквивалентно предыдущему.

2.1.2 Варианты заданий

1 Вычислить среднее арифметическое элементов, расположенных в вещественном массиве до первого и после последнего нулевых значений. Поменять местами первый и максимальный элементы, последний и минимальный элементы. Создать функции для вычисления среднего значения, определения номеров максимума и минимума.

2 Упорядочить элементы вещественного массива следующим образом: по возрастанию элементы, расположенные в массиве до минимального. Создать функции для сортировки массива и определения номеров минимального и максимального элементов.

3 Из двух целочисленных массивов различной размерности сформировать общий массив. Упорядочить полученный массив в порядке убывания его элементов. Найти наибольшие значения заданных массивов. Создать функции для сортировки массива и определения его максимального значения.

4 Из положительных значений двух целочисленных массивов различной размерности сформировать общий массив. Найти среднее арифметическое элементов, расположенных в массиве до первого простого числа. Создать функции для вычисления среднего значения и определения простого числа.

5 Из массива вещественных чисел удалить все отрицательные элементы. Найти среднее арифметическое элементов массива до и после удаления. Создать функции для удаления элемента массива и для определения среднего значения.

6 Найти наибольшее и наименьшее абсолютное значение вещественного массива. Если таких значений несколько, определить их количество. Создать функции для определения минимума и максимума.

7 Вычислить среднее значение элементов, расположенных в массиве между первым и последним нулевыми значениями. Поменять местами максимальный и минимальный элементы. Создать функции для вычисления среднего значения, определения номеров максимума и минимума.

8 Удалить из массива целых чисел все нули. Сформировать два новых массива. В первый переписать все четные элементы заданного массива, во второй – все нечетные. Упорядочить первый массив по убыванию, второй – по возрастанию. Создать функции для удаления элемента и для сортировки.

9 В массиве вещественных чисел найти элемент, превышающий его среднее арифметическое значение. Если таких элементов несколько, определить их количество. Удалить из массива первое простое число. Создать функции для определения максимума и простого числа.

10 Вычислить произведение элементов целочисленного массива, находящихся между минимальным и максимальным элементами. Создать функции для определения номеров минимального и максимального элементов.

2.2 Строки и списки

2.2.1 Неизменяемые последовательности – строки

В Python определены такие структуры данных (составные типы) как ***последовательности*** и ***отображения*** (называемые так же с***ловарями***). Словари позволяют устанавливать связи (ассоциации) ключ-значение (например, Фамилия-Адрес), поэтому с их помощью создаются так называемые ***ассоциативные массивы***.

Последовательности подразделяются на ***изменяемые*** и ***неизменяемые***. Под изменяемостью (изменчивостью) понимается возможность добавлять или удалять элементы этой последовательности, т. е. изменять количество ее элементов последовательности).

Для структур данных в Python определены функции (операции) и методы, принципиального различия между которыми нет, а есть различие синтаксическое (в правилах написания).

Для работы с текстовой информацией в Python используются ***строки*** (тип ***string***). ***Строка*** – это упорядоченная последовательность символов, используемая для хранения и представления текстовой информации. Максимальная длина строки в Python ограничивается только доступным объёмом памяти. Так что текст любого разумного размера (например, несколько тысяч страниц) может быть записан в одну строку Python. ***Строки являются неизменяемыми объектами языка.***

В Python строки и символы задаются с использованием кавычек (одиночных или двойных). Элементы (символы) в строке нумеруются, начиная с нуля. Одиночный символ является с точки зрения Python строкой, состоящей из од-

ного элемента. На рисунке 5 приведен пример инициализации строкового массива данных.

```
1  s1='Мне'
2  s2='нравится '
3  s3='Питон'
4  print(s1,s2,s3)
```

```
Run  TR
C:\archive\Python\Python3532\python.exe C:/U
Мне нравится  Питон
```

Рисунок 5 – Инициализация строкового массива

Python имеет мощный набор средств обработки строк. Информацию о всех методах можно найти с помощью интерпретатора Питон: ***help(str)***. К числу часто используемых операций и методов можно отнести:

- *Str(число)* – преобразование числа в строку;
- *S*n* – повторение строки *S n* раз;
- *S*[*i*:*j*] – извлечение среза по указанному диапазону индексов: выделение подстроки символы начиная с позиции *i* до *j-1*;
- *len*(строка) – возвращение длины строки;
- *ord*(символ) – возвращение номера символа в таблице Unicode;
- *chr*(число) – возвращение символа, соответствующего числу в таблице Unicode;
- *Str.count*('символы') – количество вхождений подстроки в строку;
- *Str.find*(‘символы’) – поиск подстроки, возвращает позицию первого совпадения, иначе – 1.

Кроме того, к строкам применимы ***операторы сравнения*** ==, !=, >, <, >=, <=. Эти операторы выполняются посимвольно слева направо до первого несовпадающего символа. Большей считается та строка, в которой больше код первого несовпадающего символа в соответствии с таблицей Unicode.

Замечание 1. Любые операции среза со строкой создают новые строки и никогда не меняют исходную строку.

Замечание 2. Если в операции среза опустить второй параметр (но поставить двоеточие), то срез берется до конца строки. Например, чтобы удалить из строки первый символ (его индекс равен 0), можно взять срез S[1:]. Аналогично если опустить первый параметр, то можно взять срез от начала строки, т. е. удалить из строки последний символ можно при помощи среза S[:-1]. Срез S[:] совпадает с самой строкой S.

Замечание 3. Если задать срез с тремя параметрами S[a:b:d], то третий параметр задает шаг, как в случае с функцией ***range***. Например, при задании значения третьего параметра, равному 2, в срез попадет каждый второй символ, а если взять значение среза, равное -1, то символы будут идти в обратном порядке. Например, можно перевернуть строку срезом S[::-1].

Пример. Различные варианты оператора извлечения среза из строки приведены на скрипте ниже (рисунок 6).

```
Python 3.6 (32-bit)
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> s='В Питоне строки неизменямы'
>>> print(s[1:-1])
 Питоне строки неизменям
>>> print(s[1:5])
 Пит
>>> print(s[2:])
Питоне строки неизменямы
>>> print(s[::4])
Вт онмм
>>> print(s[::-1])
ымянемзиен икортс енотиП В
>>> print(s[:-1])
В Питоне строки неизменям
>>>
```

Рисунок 6 – Оператор извлечения среза из строки

2.2.2 Изменяемые последовательности – списки

Списки (***mun list***) в языке Python представляют собой упорядоченную последовательность произвольных объектов. Списки являются аналогом массивов в других языках программирования, но они обладают более широкими возможностями. В отличие от строк ***списки могут содержать объекты любых типов: числа, строки и даже другие списки***. Кроме того, ***списки относятся к категории изменяемых объектов***, т. е. могут изменяться непосредственно с помощью операций присваивания, инструкций добавления и удаления элементов, методов списков.

Список создается так же, как и любая другая переменная, но инициализируется с помощью присваивания в виде разделенного запятой списка значений, заключенных в квадратные скобки. Элементы списка индексируются числовыми значениями, начиная с нуля. Обратиться к отдельному элементу списка можно, используя имя списка и индекс элемента в квадратных скобках (рисунок 7).

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> lst=[123,1.23, 'abc']
>>> print(lst)
[123, 1.23, 'abc']
>>> len(lst)
3
>>> lst[1:]
[1.23, 'abc']
>>>
```

Рисунок 7 – Обращение к элементам списка

В этом примере каждый элемент списка принадлежит к разному типу. Списки в Python поддерживают все операции над последовательностями, которые мы упоминали при обсуждении строк.

Списки могут иметь более чем один индекс, т. е. быть многомерными. Списки, содержащие три и более индекса, мало распространены, но, например, двухмерные очень часто используются для хранения пар координат X, Y. Работа с матрицами (многомерными массивами) в языке Python может быть обеспечена использованием вложенных списков. Пример извлечения элементов матрицы 3-го порядка приведен на рисунке 8.

```
Python 3.6 (32-bit)
>>>
>>> A=[[1,2,3],[4,5,6],[7,8,9]]
>>> A[1]
[4, 5, 6]
>>> A[2][2]
9
>>>
```

Рисунок 8 – Извлечение элементов матрицы 3-го порядка

С помощью первого индекса (A [1]) извлекается вторая строка, а с помощью второго (A [2] [2]) – элемент этой строки.

Пример. ***Многомерный список – список строковых величин***

Список строковых величин можно также рассматривать как многомерный, так как каждая строка сама по себе является списком символов. Следовательно, к каждому символу можно обратиться по его числовому индексу внутри определенной строки.

Пример программного кода, реализующего данную задачу, приведен на рисунке 9.

```
1   # Инициализируем список из строковых значений
2   quarter = ['Январь','Февраль',  'Март']
3   print(quarter)
4   print('Первый месяц квартала', quarter[0])
5   print('Второй месяц квартала', quarter[1])
6   print('Третий месяц квартала', quarter[2])
7
8   # Создаем многомерный список (каждый элемент - список из трех элементов)
9   coords=[[1,2,3],[4,5,6]]
10  print(coords)
11  # Выведем значения некоторых элементов внутреннего списка
12  print('Первый слева', coords[0][0])
13  print('Первый справа', coords[1][2])
14  # Выведем только один символ строковой переменной
15  print('Первый символ второго месяца квартала',quarter[1][0])

Run  TR
C:\archive\Python\Python3532\python.exe C:/Users/Кафедра/PycharmProjects/TR/T
['Январь', 'Февраль', 'Март']
Первый месяц квартала Январь
Второй месяц квартала Февраль
Третий месяц квартала Март
[[1, 2, 3], [4, 5, 6]]
Первый слева 1
Первый справа 6
Первый символ второго месяца квартала Ф
12:1  CRLF
```

Рисунок 9 – Работа с многомерным списком

2.2.3 Варианты заданий

1 В строке найдите все серии подряд идущих пробелов и замените каждую на один пробел.

2 Переставьте соседние элементы списка (A[0] с A[1], A[2] с A[3] и т. д.). Если элементов нечетное число, то последний элемент остается на своем месте.
3 Даны две строки. Удалить в первой строке первое вхождение второй строки.
4 Дан список, упорядоченный по не убыванию элементов. Определите, сколько в нем различных элементов.
5 В данной строке вставить после каждого символа 'a' символ 'b'.
6 Дан список чисел. Посчитайте, сколько в нем пар элементов, равных друг другу. Считается, что любые два элемента, равные друг другу, образуют одну пару, которую необходимо посчитать.
7 Строка состоит из слов, разделенных одним или несколькими пробелами. Найдите слово
8 Дан список. Выведите те его элементы, которые встречаются в списке только один раз. Элементы нужно выводить в том порядке, в котором они встречаются в списке.
9 В строке записано десятичное число. Запишите данное число римскими цифрами.
10Дан список чисел. Определите, сколько в этом списке элементов, которые больше двух своих соседей, и выведите количество таких элементов. Крайние элементы списка никогда не учитываются, поскольку у них недостаточно соседей.

2.3 Работа с ассоциативными массивами

2.3.1 Изменяемый тип данных – словари

Ассоциативный массив лучше всего описывается табличным представлением данных, когда каждая строка таблицы описывает характеристики какого-то объекта из множества однородных объектов (типичный пример - список учеников, их домашних телефонов и адресов). Таким образом, по значению из первого столбца такой таблицы (ключу) можно однозначно определить значения из остальных столбцов, т. е. значение ключа ассоциируется с остальными характеристиками объекта (в случае ученика – по фамилии можно найти другую информацию).

Если в ассоциативном массиве только два столбца (ключ и значение), то такой массив называется *хэш*. Такие ассоциативные массивы очень часто используются в современных информационных системах (например, пары логин-пароль).

В области моделирования процессов и явлений часто встречаются задачи, в которых значению ключа соответствует несколько параметров (например, номеру химического элемента однозначно соответствует название, атомный вес, валентность, количество протонов и пр.). В таких задачах простые *хэш*-массивы использовать уже неудобно.

Эффективный алгоритм обработки ассоциативных массивов (поиска значений, добавления и удаления значений и ключей, сортировки и пр.) в значительной степени зависит от используемого языка программирования и определённых в этом языке типов и структур данных.

В Python для ассоциативных массивов определена специальная структура данных – словарь. ***Словарь*** (тип ***dict***) – это некоторый контейнер, который может содержать несколько элементов данных в виде набора пар *ключ:значение*.

Доступ к элементам словаря осуществляется не по индексу, а по ключу. Каждому ключу в словаре соответствует некоторое значение, которое может быть ссылкой на объект любого типа, включая числа, строки, списки, множества, словари, функции и т. д. Допускается любое число уровней вложенности. Словари относятся к категории изменяемых типов.

При задании словаря используются фигурные скобки, в которых через запятую перечислены строковые пары *ключ:значение*. Строки должны быть заключены в кавычки, а между ключом и связанным с ним значением обязан стоять символ двоеточия. ***Словари относятся к категории изменяемых типов.***

Перечислим некоторые методы словарей:

- *prices.keys()* – выводит представление ключей словаря;
- *prices.values()* – выводит представление значений словаря;
- *prices.items()* – выводит представление все пар *ключ-значение*;
- *prices.copy()* – создаёт копию словаря;
- *prices.pop(key)* – возвращает значение ключа *к* и удаляет из словаря элемент с ключом *к* или *KeyError*;
- *prices.clear()* – удаляет все элементы из словаря.

Пример. Известны названия некоторых предприятий, значения планового объёма розничного товарооборота и фактического объёма розничного товарооборота. Требуется определить:

- процент выполнения плана каждым предприятием;
- количество предприятий, недовыполнивших план;
- наибольший плановый товарооборот;
- упорядочить предприятия по возрастанию планового товарооборота.

Обозначим через *k* количество предприятий и сформируем три списка: список названий предприятий (*name*), список значений планового товарооборота (*plan*), список значений фактического товарооборота (*fact*). На основании этих данных создадим список значений процентов выполнения плана (*procent*).

```
k=int(input("Введите количество предприятий: "))
name=[] # список названий предприятий (
plan=[] # список значений планового товарооборота (plan)
fact=[] # список значений фактического товарооборота (fact)

for i in range(k):
    n=input("Название: ")
    name.append(n)
    p1=int(input("План: "))
    plan.append(p1)
    p2=int(input("Факт: "))
    fact.append(p2)

# с помощью функции map() и lambda-функции создаётся список значений процентов выполнения плана
procent=map(lambda x,y :x*100/y, fact, plan)  # список значений процентов выполнения плана

# с помощью функции zip() формируется два итоговых ассоциативных массива (словаря)
faktdict=dict(zip(name, procent))
plandict=dict(zip(plan, name))

print (20* "=")
print ("Процент выполнения плана каждым предприятием:")
noplan=0
for key in faktdict:
    s=faktdict[key]
    if s < 100:
        noplan+=1
    print (key,": ", s)
print("Количество предприятий, не выполнивших план: ", noplan)
print ("Наибольший плановый товарооборот: ", max(plan))

print (20* "=")
print('Предприятия по возрастанию плана')

b = plandict.keys() # получаем список ключей
b = list(b)         # превращаем его в обычный список
b.sort()            # сортируем список
for key in b:
    print(plandict[key],':',key )  # вывод элементов словаря ( значение-ключ) по возрастанию ключа (плана)
```

Рисунок 10 – Обработка списка и словаря

Количество предприятий, недовыполнивших план, будем определять в результате сравнения процента выполнения со 100 процентами в цикле по всем предприятиям. Пример программного кода, реализующего данную задачу, приведен на рисунке 10.

Здесь с помощью функции ***map()*** и одноразовой ***lambda***-функции создаётся список процентов выполнения плана и с помощью функции ***zip ()*** формируется два итоговых ассоциативных массива. Сортировка таких ассоциативных массивов производится по первому столбику, поэтому важен порядок аргументов в функции ***zip ()***, а также порядок индексов при выводе результатов.

Результат выполнения программы приведен на рисунке 11.

```
C:\Users\Таня\AppData\Local\Programs\Python\Python3
Введите количество предприятий: 3
Название: фирма1
План: 230
Факт: 120
Название: фирма2
План: 200
Факт: 239
Название: фирма3
План: 300
Факт: 350
====================
Процент выполнения плана каждым предприятием:
фирма3 :  116.66666666666667
фирма2 :  119.5
фирма1 :  52.17391304347826
Количество предприятий, не выполнивших план:  1
Наибольший плановый товарооборот:  300
====================
Предприятия по возрастанию плана
фирма2 : 200
фирма1 : 230
фирма3 : 300
```

Рисунок 11 – Результат выполнения программы

2.3.2 Варианты заданий

1 В словаре содержится информация о зимней сессии 3-го курса. Сведения о каждом студенте содержат следующие данные: *фамилию, номер группы* (от 1 до 7), *оценки по двум предметам*. Напишите программу, которая вводит эту информацию и печатает следующие данные: фамилии студентов, имеющих задолженность хотя бы по одному предмету; процент студентов, сдавших все экзамены на 5 и 4.

2 В словаре содержится информация о зимней сессии. Сведения о каждом студенте содержат следующие данные: *фамилию, номер курса, оценки по двум предметам*. Напишите программу, которая вводит эту информацию и печатает следующие данные: фамилии студентов, сдавших оба предмета на 5; среднюю успеваемость студентов каждого курса.

3 Сведения о студентах вуза записаны в массиве и содержит следующую информацию: *фамилия, пол (М или Ж), возраст, курс*. Напишите программу, которая вводит эту информацию и печатает следующие данные: номер курса, на котором наибольший процент мужчин; фамилии (в алфавитном порядке) студенток 5-го курса.

4 Багаж пассажира характеризуется *количеством вещей и общим весом вещей*. Пусть дан словарь, содержащий информацию о багаже нескольких пассажиров. Найдите багаж, средний вес каждой вещи которого отличается не более чем на 0.3 кг от среднего веса вещей всех пассажиров; определите, имеются ли два пассажира, багажи которых совпадают по числу вещей и отличаются по весу не более чем на 0,5 кг.

5 Багаж пассажира характеризуется *количеством вещей и общим весом вещей*. Пусть дан словарь, содержащий информацию о багаже нескольких пассажиров. Найдите число пассажиров, имеющих более двух вещей, и число пассажиров, количество вещей которых превосходит среднее число вещей; выясните, имеется ли пассажир, багаж которого превышает багаж каждого из остальных пассажиров и по количеству вещей, и по весу.

6 Багаж пассажира характеризуется *количеством вещей и общим весом вещей*. Пусть дан словарь, содержащий информацию о багаже нескольких пассажиров. Выясните, имеется ли пассажир, багаж которого состоит из одной вещи весом не менее 30 кг; удалите сведения о багаже, общий вес вещей в котором меньше, чем 10 кг.

7 Пусть дан массив, содержащий сведения о книгах. Сведения о каждой книге – это *фамилия автора, название и год издания*. Найдите названия книг данного автора, изданных с 2010 года; определите, имеется ли книга с названием «Информатика»; если да, то сообщите фамилию автора и год издания книги, если таких книг несколько, то сообщите имеющиеся сведения обо всех этих книгах.

8 Сведения о каждой машине включают в себя следующую информацию: *номер, модель, цвет, сведения о владельце, год последнего техосмотра*. Выберите данные обо всех машинах, не прошедших техосмотр в текущем году; напечатайте информацию обо всех владельцах «Ауди» черного цвета.

9 Сведения о каждой машине включают в себя следующую информацию: *модель, номер, цвет, сведения о владельце, год последнего техосмотра*. Выберите данные обо всех машинах, прошедших техосмотр в текущем году; по фамилии владельца машины выдайте всю имеющуюся информацию о ней.

10 Пусть дан словарь, содержащий сведения о жителях, обслуживаемых данной поликлиникой. Сведения содержат следующую информацию: *фамилию, адрес, наличие прививки от дифтерии, год прохождения последней флюорографии*. Напечатайте фамилии и адреса тех жильцов, которые не сделали прививку от дифтерии; сформируйте массив с данными о жильцах, у которых на данный момент просрочена дата флюорографии.

Задание 3. Prolog: элементы логического программирования

Язык программирования Prolog является универсальным инструментальным средством для изучения основ искусственного интеллекта и логического программирования.

Студенты должны хорошо усвоить следующие понятия: предметная и проблемная область; представление объекта и его свойств с помощью констант и переменных PROLOG; предикат, его синтаксис и семантика; описание отношений с помощью предикатов; виды утверждений: факт, правило, запрос - их синтаксис и семантика; алгоритм работы интерпретатора; рекурсивное описание процессов, данных и отношений; сложные структуры данных.

3.1 Поиск с возвратом

3.1.1 Предикаты *fail* и отсечения

Fail – это тождественно-ложный предикат, искусственно создающий ситуацию неуспеха. После выполнения этого предиката управление передается в точку отката и поиск продолжается. Использование предиката ***fail*** позволяет найти все решения задачи.

Чтобы ограничить пространство поиска и прервать поиск решений при выполнении какого–либо условия, используется ***предикат отсечения*** (обозначается символом «***!***»). Пройдя через отсечение один раз, невозможно вернуться назад, так как этот предикат является тождественно-истинным. Процесс может только перейти к следующей подцели, если таковая имеется. Например, *p:- p1, p2, !, p3*. Если достигнуты цели *p1* и *p2*, то возврат к ним для поиска новых решений невозможен.

Пример. База данных содержит факты вида: ***student(имя, курс).*** Создать проект, позволяющий сформировать список студентов 1-го курса.

Пример программного кода и результат выполнения программы приведены на рисунках 12-13.

```
PREDICATES
student(symbol,integer)
spisok
CLAUSES
student(vova,3).
student(lena,1).
student(dima,1).
student(ira,2).
student(marina,1).
spisok:-student(X,1),write(X),nl,fail.
GOAL
write("Список студентов 1-курса"),nl,spisok.
```

Рисунок 12 – Prolog: создание списка студентов

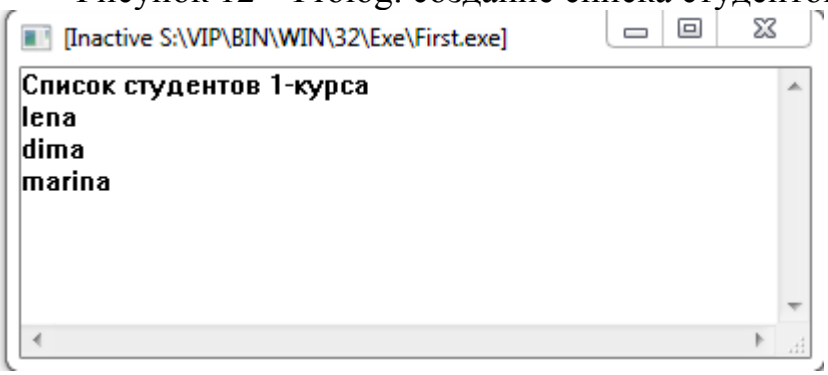

Рисунок 13 – Результат выполнения программы: список студентов

Пример. Создать проект, реализующий железнодорожный справочник. В справочнике содержится следующая информация о каждом поезде: *номер поезда, пункт назначения и время отправления*.

а) вывести всю информацию из справочника.

Пример программного кода и результат выполнения программы приведены на рисунках 14-15.

```
DOMAINS
nom=integer
p, t=string
PREDICATES
poezd(nom,p,t)
CLAUSES
poezd(233,moskva,"12-30").
poezd(257,moskva,"22-40").
poezd(133,armavir,"10-20").
poezd(353,armavir,"20-40").
poezd(353,adler,"02-30").
poezd(413,adler,"11-10").
poezd(256,piter,"21-30").
GOAL
write("      Расписание поездов"), nl,
write("Номер Пункт прибытия Время отправления"),
nl, poezd(N,P,T), write(N,"   ",P,"    ",T),nl,fail.
```

Рисунок 14 – Prolog: вывод информацию из справочника

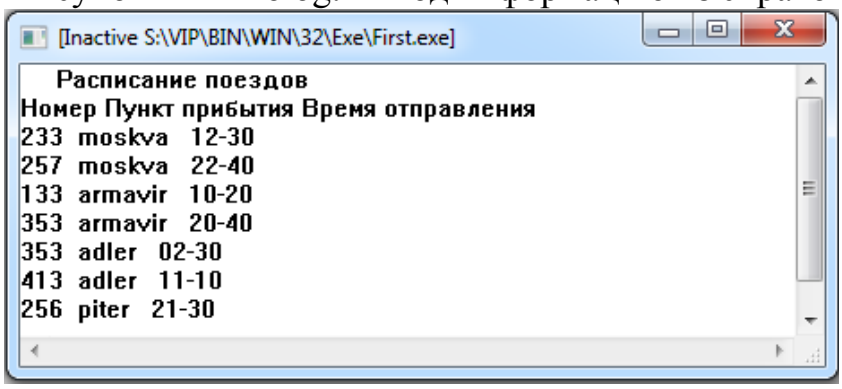

Рисунок 15 – Результат выполнения программы: список поездов из справочника

б) организовать поиск поезда по пункту назначения.

Пример программного кода приведен на рисунке 16.

```
GOAL
write (" Пункт назначения:"), Readln(P), nl,
write ("Номер     Время отправления"), nl,
poezd(N,P,T), write(N,"                    ",T), nl, fail.
```

Рисунок 16 – Prolog: поиск поезда по пункту назначения

Здесь *Readln* – стандартный предикат ввода строкового значения. Результат выполнения программы приведен на рисунке 17.

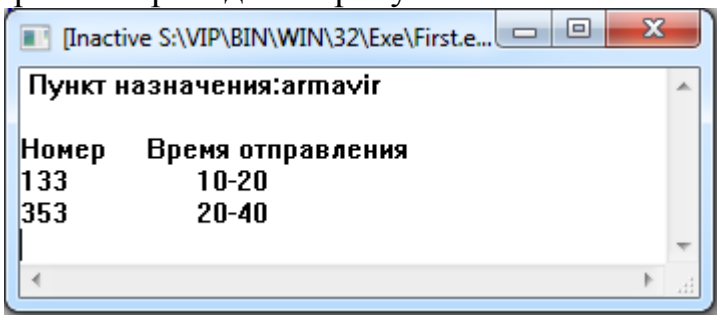

Рисунок 17 – Результат выполнения программы: список поездов из справочника

в) вывести информацию о поездах, отправляющихся в заданный временной промежуток.

Пример программного кода и результат выполнения программы приведены на рисунках 18-19.

```
GOAL
write(" Время отправления:"),nl,
write("с..."), Readln(T1),
write("go..."), Readln(T2), nl,
write("Номер  Пункт назначения  Время отправления"),
nl,poezd(N,P,T),T>=T1,T<=T2,write(N,"  ",P,"  ",  T),
nl, fail.
```

Рисунок 18 – Prolog: вывод информации о поездах

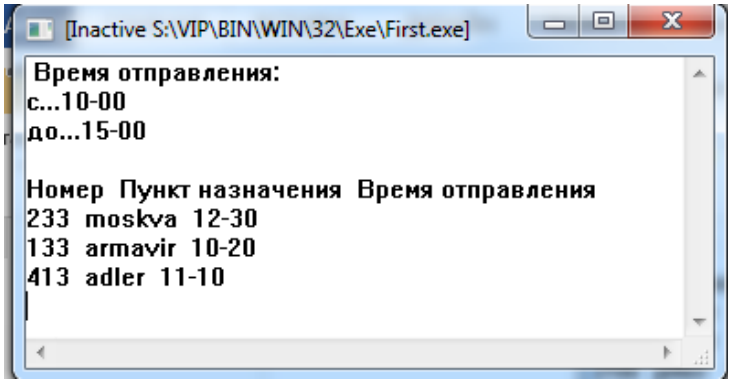

Рисунок 19 – Результат выполнения программы: вывод информации о поездах из справочника

3.1.2 Варианты заданий

1 База данных содержит факты вида: *отдыхает (имя, город), Испания (город), Россия (город), женщина (имя), мужчина (имя)*.

а) вывести список женщин, отдыхающих в России;

б) вывести список мужчин, отдыхающих в Испании.

2 База данных содержит факты вида: *книга (автор, название, издательство, год_издания), Россия (город)*.

а) вывести весь список книг;

б) вывести список книг авторов Пушкина и Чехова;

в) вывести список книг, изданных в издательстве «Москва» не ранее 2016 года.

3 Составить программу, реализующую авиасправочник. В справочнике содержится следующая информация о каждом рейсе*: номер рейса, пункт назначения, время вылета, дни (ежедн., чет, нечет)*.

Вывести:

а) всю информацию из справочника;

б) информацию о самолетах, вылетающих в заданный пункт по четным дням;

в) информацию о самолетах, вылетающих ежедневно не позже указанного времени.

4 Составить программу, реализующую географический справочник. В справочнике содержится следующая информация о каждой стране: *название страны, название столицы, численность населения, географическое положение (Европа или Азия)*.

Вывести:

а) всю информацию из справочника;

б) информацию о странах, численность населения которых превышает заданное значение;
в) информацию о европейских странах, численность населения которых не превышает заданное значение.

5 Составить программу, реализующую словарь. В словаре содержится следующая информация: *слово и его перевод (русские и английские слова)*. Реализовать вывод всего словаря, перевод с русского на английский, с английского на русский (с несколькими значениями).

6 Составить программу, реализующую телефонный справочник. В справочнике содержится следующая информация о каждом абоненте: *имя и телефон*. Реализовать вывод всей информации из справочника, поиск телефона по имени, поиск имени по телефону.

7 База данных содержит факты вида: *ученик (имя, класс) и увлекается (имя, хобби)*. Составить программу, которая выводит:
а) список всех учеников и их увлечения;
б) подбирает одному из учеников указанного класса, увлекающемуся кино, пару из других классов. Вывести все возможные пары.

8 База данных содержит факты вида: *ученик (имя, класс) и играет (имя, вид спорта)*. Составить программу, которая:
а) выводит список всех учеников заданного класса и вид спорта, которым они увлекаются;
б) подбирает одному из учеников указанного класса, играющему в бадминтон, пару из других классов. Вывести все возможные пары.

9 Создать проект, реализующий железнодорожный справочник. В справочнике содержится следующая информация о каждом поезде: *номер поезда, пункт назначения и время отправления*.
а) вывести всю информацию из справочника;
б) организовать поиск поезда по пункту назначения.

10 База данных содержит факты вида: *student (имя, курс) и играет (имя, инструмент)*. Составить программу, которая:
а) выводит список студентов 3-го курса и музыкальный инструмент, на котором каждый из них играет;
б) реализует поиск имени по музыкальному инструменту.

3.2 Рекурсия

3.2.1 Рекурсивная функция

Рекурсивная функция – это функция, которая вызывает сама себя либо непосредственно, либо с помощью другой функции. Функция вызывает сама себя до тех пор, пока не будет соблюдено некоторое условие. Такое условие называют ***граничным***.

Рекурсивные функции обладают многими достоинствами и являются мощным средством при создании оптимальных программ в таких задачах, как перевод с языка на язык, в том числе при трансляции, в доказательствах теорем и др.

Рекурсивное правило всегда состоит, по крайней мере, из двух частей, одна из которых является не рекурсивной. Она определяет граничное условие.

Пример. Рекурсивное вычисление факториала (рисунок 20)

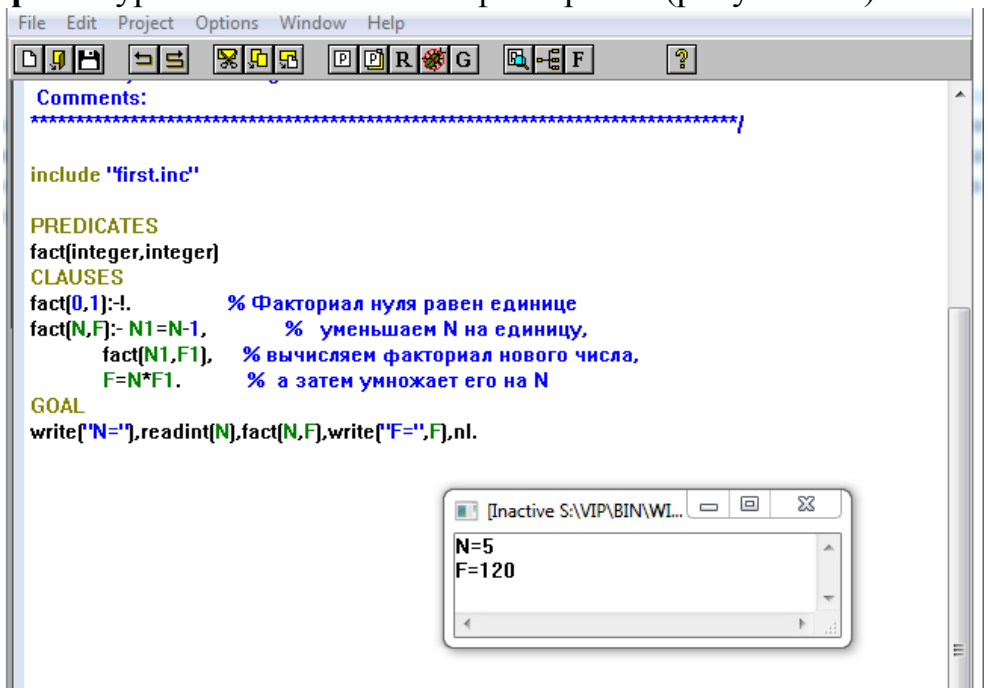

Рисунок 20 – Рекурсивное вычисление факториала

3.2.2 Варианты заданий

1 Подсчитать, сколько раз встречается некоторая буква в строке. Строка и буква должны вводиться с клавиатуры. Для разделения строки на символы использовать стандартный предикат *frontchar(String, Char, StringRest)*, позволяющий разделять строку *String* на первый символ *Char* и остаток строки *StringRest*.

2 Вычислить значение *n*-го члена ряда Фибоначчи: $f(0)=0$, $f(1)=1$, $f(n)=$
$=f(n-1)+f(n-2)$.

3 Вычислить произведение двух целых положительных чисел (используя суммирование).

4 Определить, сколько раз встречается некоторое слово в строке. Строка и слово должны вводиться с клавиатуры. Для разделения строки на слова использовать стандартный предикат *fronttoken(String, Lexeme, StringRest)*, позволяющий разделить строку *String* на первое слово *Lexeme* и остаток строки *StringRest*.

5 Поменять порядок следования букв в слове на противоположный. Для разделения строки на символы использовать стандартный предикат *frontchar (String, Char, StringRest)*, позволяющий разделять строку *String* на первый символ *Char* и остаток строки *StringRest*.

6 Вычислить сумму ряда целых нечетных чисел от *1* до *n*.

7 Поменять порядок следования слов в предложении на противоположный. Для разделения строки на слова использовать стандартный предикат *fronttoken (String, Lexeme, StringRest)*, позволяющий разделить строку *String* на первое слово *Lexeme* и остаток строки *StringRest*.

8 Вычислить сумму ряда целых четных чисел от *2* до *n*.

9 Организовать ввод целых положительных чисел и их суммирование до тех пор, пока сумма не превысит некоторого порогового значения. Введенные отрицательные целые числа суммироваться не должны.

10 Организовать ввод букв и их соединение в строку до тех пор, не будет введен символ #. Для присоединения символа к строке использовать стандартный предикат *frontchar (String, Char, StringRest*), позволяющий присоединять символ *Char* к строке *StringRest* и получать строку *String*.

3.3 Программирование логических функций

3.3.1 Определение истинности логической формулы

Процесс программирования достаточно простых структур данных очень удобно разобрать на примере определения истинности логической формулы. В таблице 1 приведены восемь логических функций, для каждой из которых дается таблица истинности, наиболее распространенное символическое обозначение и предикат.

Пример. Описание логической функции «штрих Шеффера» с использованием трехместного предиката **sht** (* ,* ,*) .

В разделе **domains** опишем аргументы предиката **sht** (*,*,*) как объекты типа **char.** Поместим в раздел **predicates** предикат **sht (c, c, c)** , где первый и второй аргументы **c** предиката **sht** описывают операнды *A, B,* а третий **c** – соответствующее им значение функции из таблицы 1. Раздел *clauses* будет содержать базу знаний функции «штрих Шеффера» из таблицы 1.

Таблица 1. Таблица истинности логических функций

№	Функция	Символическое обозначение	Таблица истинности				Предикат
			$A = T$, $B = T$	$A = T$, $B = F$	$A = F$, $B = T$	$A = F$, $B = F$	
1	Конъюнкция	$A \wedge B$	T	F	F	F	con(•,•,•)
2	Эквивалентность	$A \sim B$	T	F	F	T	eqv(•,•,•)
3	Дизъюнкция	$A \vee B$	T	T	T	F	diz(•,•,•)
4	Отрицание	$\overline{A}$	F	F	T	T	neg(•,•)
5	Импликация	$A \rightarrow B$	T	F	T	T	imp(•,•,•)
6	Стрелка Пирса	$A \downarrow B$	F	F	F	T	str(•,•,•)
7	Штрих Шеффера	$A \mid B$	F	T	T	T	sht(•,•,•)
8	Неэквивалентность	$A \oplus B$	F	T	T	F	neqv(•,•,•)

Prolog – программа, описывающая логическую функцию, приведена на рисунке 21.

```
/******************************
DOMAINS
c=char
PREDICATES
sht(c,c,c)
CLAUSES
   sht('T','T','F').
   sht('T','F','T').
   sht('F','T','F').
   sht('F','F','T').
```

Рисунок 21 – Prolog – программа: описание логической функции

Целью (одной из четырех возможных) может быть предикат *sht(‘**F**’ ,’**F**‘ , **X**)*, в котором требуется определить значение ***X*** логической функции «штрих Шеффера» при значениях *A =’**F**’ **и** B =’**F**’*.

Пример. Рассмотрим программирование на Prolog логических формул, используя в качестве примера формулу:

$$\overline{A \wedge B} \vee C.$$

Для описания этой формулы требуются базы данных, содержащие предикаты отрицания *neg (*,*)*, конъюнкции *con (* ,* ,*)* и дизъюнкции *diz (* ,* , *)*, а также правило с головой – предикатом *pred (A, B, C, Z)* и телом, состоящим из предикатов *con*, *neg* и *diz*. Логические операции в данной формуле выполняются в следующем порядке: конъюнкция, отрицание, дизъюнкция.

Конъюнкцию $A \wedge B = X$ реализует предикат *con(A, B, X)*, отрицание $\overline{X} = Y$ – предикат *neg(X, Y)* и дизъюнкцию $Y \vee C = Z$ – предикат *diz(Y, C, Z)* . В итоге правило примет вид:

pred(A,B,C,Z):-con (A, B, X), neg (X, Y), diz (Y, C, Z).

На основании полученного правила и баз знаний, описывающих логические функции отрицания ***neg** (*,*)*, конъюнкции ***con** (* ,* ,*)* и дизъюнкции ***diz** (* ,*,*)* из таблицы 1, составляется программа, представленная на рисунке 22, для определения истинности логической формулы.

```
/*********************************************
DOMAINS
  c=char
PREDICATES
  con(c,c,c)
  diz(c,c,c)
  neg(c,c)
  pred(c,c,c)
CLAUSES
  con('T','T','T'). % База данных конъюнкции
  con('T','F','F').
  con('F','T','F').
  con('F','F','F').
  diz('T','T','T'). % База данных дизъюнкции
  diz('T','F','T').
  diz('F','T','T').
  diz('F','F','F').
  neg('F','T').     % База данных отрицания
  neg('T','F').
  pred(A,B,C,Z):-
  con(A,B,X),neg(X,Y),diz(Y,C,Z).
```

Рисунок 22 – Программирование логической функции

Одной из целей (их всего 8) может быть предикат *pred (‘T’ , ‘F’ , ‘T’ , ‘Z’)* , в котором при значениях *A = ‘T’, B = ‘F’, C =’T’* определяется итоговое значение *Z* логической формулы.

Программа на Prolog с указанной целью выполняется следующим образом. Цель сопоставляется с головой правила, и его переменные *A, B, C* получают соответствующие значения *‘T‘, ‘F’, ‘T’* которые передаются в предикаты тела правила

con(‘T’, ‘F’, X) и diz (Y, ‘T’, Z).

Первый предикат – цель *con (‘T’, ’F’, X)* в базе знаний Prolog находит сопоставимый предикат *con (‘T‘, ‘F', 'F')*, в результате чего *X =’F’*. Второй предикат – цель *neg(X, Y) с X = ‘F’* в базе данных Prolog находит сопоставимый предикат *neg(‘F’, ‘T’)*, в результате чего получим *Y = ‘T’*.

Третий предикат – цель *diz(Y, C, Z)* с *Y = ‘T’* и *C = ‘T’* в базе знаний находит сопоставимый предикат *diz(‘T’, ’T’, ‘T’)*, в результате чего получаем конечное значение логической формулы *Z =’T’*, которое передается в голову правила и выводится как ответ *Z = ‘T’*.

3.3.2 Варианты заданий

Составьте на Prolog программы логических формул и определите их таблицы истинности для всех наборов значений логических переменных *A, B, C*.

1	$(A\|B)\vee\overline{A\vee C}$	**2**	$(A\downarrow B)\vee(B\to C)$
3	$(A\oplus B)\wedge(\overline{B}\sim C)$	**4**	$(A\vee B)\downarrow(A\vee\overline{C})$
5	$(A\|B)\vee\overline{A\wedge C}$	**6**	$A\vee\overline{B\oplus C}$
7	$\overline{A}\wedge(A\to(B\to C))$	**8**	$(A\vee\overline{B})\wedge(B\to C)$
9	$(A\vee\overline{B})\oplus(B\downarrow C)$	**10**	$(A\downarrow B)\to\overline{B\wedge C}$

Задание 4 Вопросы для самостоятельного изучения

4.1 Язык программирования Python (базовый уровень)

1 История и тенденции развития языков программирования.

2 Области применения языка программирования Python.

3 Переменные в Python. Наименование. Модель памяти Python при работе с переменными.

4 Функции в Python. Создание функций.

5 Создание программ на языке Python в отдельном файле. Отличие от интерактивного режима.

6 Строки и операции над строками.

7 Операторы отношений в Python. Логические операции над объектами.

8 Условная инструкция *if*.

9 Модули.

10 Создание собственных модулей.

11 Строковые методы. Отличие функций от методов.

12 Списки. Создание списка.
13 Операции над списками.
14 Псевдонимы и клонирование списков.
15 Методы списка.
16 Преобразование типов в Python (списки, строки).
17 Вложенные списки.
18 Циклы.
19 Цикл *for* для списков и строк.
20 Функция *range()* и цикл *for*.
21 Способы генерации списка.
22 Цикл *while*.
23 Вложенные циклы (на примере вложенных списков)/
24 Множества и операции над ними.
25 Кортежи и операции над ними.
26 Словари и операции над ними.
27 Обработка исключений.
28 Работа с файлами. Менеджер контекста.
29 Объектно-ориентированное программирование. Классы, объекты.
30 Иерархия наследования (класс *object*).
31 Полиморфизм.
32 Структура оконного приложения на примере модуля *tkinter* (обработка событий).
33 Реализация шаблона «Модель-вид-контроллер» на примере модуля *tkinter*.
34 Структура клиент-серверного приложения (модуль *socket*).

4.2 Язык программирования Prolog (базовый уровень)

1 Понятие декларативного программирования. Назначение декларативных языков, их особенности.
2 Предикаты как отношения между объектами. Структура предикатов.
3 Prolog – язык логического программирования. Фундаментальные свойства Prolog.
4 Структура Prolog - программы.
5 Отношения-факты и отношения-правила: форма записи, примеры.
6 Запросы: внешние и внутренние, простые и составные.
7 Объекты данных: атомы и числа, переменные, структуры.
8 Механизм поиска с возвратом в Prolog.
9 Стандартный предикат fail: назначение, примеры использования.
10 Предикат отсечения: назначение, примеры использования.
11 Арифметика Prolog: операции, функции, примеры использования.
12 Рекурсивная организация вычислений.
13 Предикаты ввода: назначение, виды, примеры использования.
14 Понятие списка.

15 Стандартные задачи обработки списков: генерирование списка.
16 Стандартные задачи обработки списков: объединение списков.
17 Стандартные задачи обработки списков: поиск элемента в списке.
18 Стандартные задачи обработки списков: удаление элементов списка.
19 Стандартные задачи обработки списков: вставка элементов в список.
20 Динамические базы данных. Объявление динамической базы данных.
21 Работа с фактами динамической базы данных.
22 Стандартные предикаты для работы со строками.

Список литературы

1 Городняя Л. В. Основы функционального программирования: курс лекций : учебное пособие для студентов вузов, обучающихся по специальности 351400 «Прикладная информатика». – Москва : Интернет-Университет информационных технологий, 2004. – 272 с.

2 Стобо Дж. Язык программирования Пролог.- Москва : Радио и связь, 1993. – 368 с.

3 Марк Саммерфилд. Python на практике : пер. с англ. – Москва : ДМК Пресс, 2014. – 338 с.

4 Дэвид М. Бизли. Python. Подробный справочник – 4-е изд.: пер. с англ. – Санкт–Петербург : Символ–Плюс, 2010. — 864 с.

5 Марк Саммерфилд. Программирование на Python 3. Подробное руководство : пер. с анг. – Санкт–Петербург : Символ-Плюс, 2009. – 608 с.

6 Сузи Р. А. Язык программирования Python : учеб. пособие. – Москва : ИНТУИТ, БИНОМ. Лаборатория знаний, 2006. — 328 с.

7 Дмитриева Т. А., Цуканова Н. И. Теория и практика логического программирования на Visual Prolog 7 : учеб. пособие. Москва : Горячая линия – Телеком, 2011. – 232 с.

8 Адаменко А., Кучуков А. Логическое программирование и Visual Prolog (с CD). – Санкт–Петербург : БХВ-Петербург, 2003. – С. 990.

9 И. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG = Prolog Programming For Artificial Intelligence. – Москва : Вильямс, 2004. – 640 с.

10 Сайт, содержащий необходимые дистрибутивы и полную информацию для языка программирования Python. Интерпретатор для Python можно использовать как программируемый высокоуровневый калькулятор.
URL: https://www.python.org/ .

11 Сайт, посвященный свободно распространяемому пакету Sympy, представляющему собой библиотеку Python символьных вычислений.
URL: http://sympy.org/ .

12 Электронный учебник (свободный доступ), содержащий полную информацию о языке программирования Python.
URL: http://docs.python.org/3/tutorial/index.html/ .

13 Основы программирования на языке Visual Prolog .URL: http://www.intuit.ru/studies/courses/12333/1180/info .
14 Логическое программирование на языке Visual Prolog. URL: http://www.ict.edu.ru/ft/005567/soldatova_lezina.pdf/ .

СОДЕРЖАНИЕ

Введение 3

Задание 1. Функциональное и логическое программирование 3
1.1 Варианты заданий 3
Задание 2. Python: элементы функционального программирования 4
2.1 Функции 4
2.1.1 Элементы функционального программирования в Python 4
2.1.2 Варианты заданий 8
2.2 Строки и списки 9
2.2.1 Неизменяемые последовательности – строки 9
2.2.2 Изменяемые последовательности – списки 11
2.2.3 Варианты заданий 12
2.3 Работа с ассоциативными массивами 13
2.3.1 Изменяемый тип данных – словари 13
2.3.2 Варианты заданий 16
Задание 3. Prolog: элементы логического программирования 17
3.1 Поиск с возвратом 18
3.1.1 Предикаты *fail* и отсечения 18
3.1.2 Варианты заданий 20
3.2 Рекурсия 21
3.2.1 Рекурсивная функция 21
3.2.2 Варианты заданий 22
3.3 Программирование логических функций 23
3.3.1 Определение истинности логической формулы 23
3.3.2 Варианты заданий 25
Задание 4 Вопросы для самостоятельного изучения 25
4.1 Язык программирования Python (базовый уровень) 25
4.2 Язык программирования Prolog (базовый уровень) 26
Список литературы 27

Змызгова Татьяна Рудольфовна

ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Методические указания и контрольные задания
к выполнению контрольной работы
по дисциплине «Функциональное и логическое программирование»
для студентов очной и заочной формы обучения направления подготовки
09.03.04 «Программная инженерия»

Редактор Г.В. Меньщикова

Подписано в печать 16.10.17	Формат 60*84*1/16	Бумага 65 г/м^2
Печать цифровая	Усл. печ. л. 2,0	Уч.-изд. л. 2,0
Заказ №168	Тираж 25	Не для продажи

БИЦ Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.