Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Севастопольский государственный университет»

ОТЧЕТ

о выполнении лабораторной работы № 2 по дисциплине

«Методы и системы искусственного интеллекта»

Выполнил:

ст. гр. ИС/б-22-1-о

Гюнтер М. Ю .

Проверил:

доцент кафедры

“Информационные системы”

Бондарев В. Н.

Севастополь,2025

# Цель работы: Исследование неинформированных методов поиска решений задач в пространстве состояний, приобретение навыков программирования интеллектуальных агентов, планирующих действия на основе методов слепого поиска решений задач.

# Ход работы

## Выполнение задания 1

Продемонстрируем работу метода str.split() (рисунок ‎2.1), возвращающего список слов на основании строки и разделителя с заданным ограничением, по умолчанию равного пробелу, с помощью кода, приведённого в листинге ‎2.1.

Листинг ‎2.1 – Демонстрация метода str.split()

nums\_str = '0 1 2 3 4 5 6 7 8 9'

nums\_int = nums\_str.split()

nums\_inc = [int(num) + 1 for num in nums\_int]

for num in nums\_inc:

print(num, end = " ")



Рисунок ‎2.1 – Результат обработки строки с использованием str.split()

Также продемонстрируем работу аналогичного метода str.join() (рисунок ‎2.2), объединяющего итерируемый объект на основании строки соединителя, для которой вызывается метод, с помощью кода, приведённого в листинге ‎2.2.

Листинг ‎2.2 – Демонстрация метода str.join()

nums\_str\_comma = '0, 1, 2, 3, 4, 5, 6, 7, 8, 9'

print(f'Оригинальная строка: {nums\_str\_comma}')

nums\_int = nums\_str\_comma.split(', ')

nums\_str\_space = " ".join(nums\_int)

print('Строка с изменённым разделителем: {}'.format(nums\_str\_space))



Рисунок ‎2.2 – Изменение разделителя строки с использованием str.join()

Продемонстрируем работу метода str.count() (рисунок ‎2.3), возвращающего количество неперекрывающихся повторений заданной подстроки в строке, с помощью кода, приведённого в листинге ‎2.3.

Листинг ‎2.3 – Демонстрация метода str.count()

haystack = 'Хороший код — это читаемый код.'

needle = 'код'

count = haystack.count(needle)

print(f'''Сколько раз слово "{needle}" содержится в предложении "{haystack}"?\n{count}''')



Рисунок ‎2.3 – Результат поиска подстроки с использованием str.count()

Продемонстрируем работу метода str.startswith() (рисунок ‎2.4), возвращающего True, если строка начинается с заданной подстроки, и False, если строка не начинается с заданной подстроки, с помощью кода, приведённого в листинге ‎2.4.

Листинг ‎2.4 – Демонстрация метода str.startswith()

fruits = [

"Apple",

"banana",

"apricot",

"Grape",

"avocado",

"orange",

"kiwi",

"Almond",

"Blueberry",

"artichoke"

]

count = 0

for i in fruits:

if i.lower().startswith('a'):

count += 1

print('''Количество фруктов, название которых начинается с буквы "a": \n%d''' % count)



Рисунок ‎2.4 – Результат поиска фруктов на букву “a” с использованием str.startswith()

Продемонстрируем работу метода str.endswith() (рисунок ‎2.5), возвращающего True, если строка заканчивается заданной подстрокой, и False, если строка не заканчивается заданной подстрокой, с помощью кода, приведённого в листинге ‎2.5.

Листинг ‎2.5 – Демонстрация метода str.endswith()

sentences = [

"Синтаксис Python очень читаемый.",

"Отладка кода требует терпения",

"Алгоритмы — это основа программирования.",

"Компилятор преобразует код в машинный язык",

"Тестирование необходимо для качества продукта.",

"Версионирование кода с помощью Git обязательно",

"Функции помогают избежать повторения кода.",

"Базы данных хранят информацию структурированно",

"Рекурсия может быть сложной для понимания.",

"Веб-разработка включает фронтенд и бэкенд"

]

count = 0

for i in sentences:

if i.endswith('.'):

count += 1

print('Число предложений, где пропущена точка: \n%d' % count)



Рисунок ‎2.5 – Результат поиска предложений, где пропущена точка, с использованием str.endswith()

Продемонстрируем работу оставшихся методов обработки строк str.format(), str.split(), str.lsplit(), str.rsplit(), str.capitalize(), str.title(), str.index(), str.rindex(), str.replace(), str.rsplit(), str.partition() и str.rpartition() (рисунок ‎2.6), с помощью кода, приведённого в листинге ‎2.6.

Листинг ‎2.6 – Демонстрация оставшихся методов обработки строк

# format method

print('Это пример {} текста с помощью метода {}.'.format('форматирования', 'format()'))

# strip method

print(' Это пример удаления пробелов в начале и конце строки '.strip())

# lstrip method

print(' Это пример удаления пробелов в начале строки'.lstrip())

# rstrip method

print('''Это пример удаления символов "\_abc" в конце строки\_abc\_abc\_abc\_abc'''.rstrip('\_abc'))

# capitalize method

print('эТО ПРИМЕР ИЗМЕНЕНИЯ СТРОКИ, ПРИ КОТОРОМ ТОЛЬКО ЕЁ ПЕРВЫЙ СИМВОЛ ПРОПИСНОЙ'.capitalize())

# title method

print('ЭТО ПРИМЕР ИЗМЕНЕНИЯ СТРОКИ, ПРИ КОТОРОМ ТОЛЬКО ПЕРВЫЙ СИМВОЛ КАЖДОГО ЕЁ СЛОВА ПРОПИСНОЙ'.title())

# index method

print("Индекс подстроки \"ab\" в строке \"bc ab ab ab ab ab ab bc\" с начала строки: %d" % "bc ab ab ab ab ab ab bc".index("ab"))

# rindex method

print("Индекс подстроки \"ab\" в строке \"bc ab ab ab ab ab ab bc\" с конца строки: %d" % "bc ab ab ab ab ab ab bc".rindex("ab"))

# replace method

print('Это dummy замены подстроки в строке'.replace('dummy', 'пример'))

# rsplit method

print('''Это пример разделения строки "1, 2, 3, 4, 5, 6" с конца с ограничением 4:''')

print('1, 2, 3, 4, 5, 6'.rsplit(", ", 4))

# partition method

print('''Это пример раделения строки "abc|def|ghi на 3 части слева с разделителем "|":"''')

print('abc|def|ghi'.partition('|'))

# rpartition method

print('''Это пример разделения строки "abc|def|ghi" на 3 части справа с разделителем "|":''')

print('abc|def|ghi'.rpartition('|'))

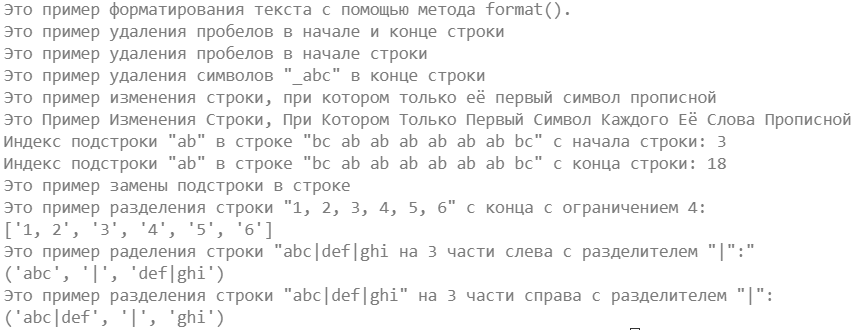


Рисунок ‎2.6 – Результат работы оставшихся методов обработки строк

## Выполнение задания 2

Продемонстрируем работу методов обработки списков list.append(), list.extend(), list.insert(), list.pop(), list.remove(), list.reverse(), list.sort(), list.count(), list.index() (рисунок ‎2.7) с помощью кода, приведенного в листинге ‎2.7.

Листинг ‎2.7 – Демонстрация методов обработки списков

# append method

myList = [1, 2, 3]

print('Исходный список: ', myList)

myList.append(4)

print('Список с единственным добавленным элементом: ', myList, end="\n\n")

# extend method

myList = [1, 2, 3]

print('Исходный список: ', myList)

myList.extend([4, 5, 6])

print('Список с несколькими добавленными элементами: ', myList, end="\n\n")

# insert method

myList = [1, 2, 3, 5, 6, 7]

myNum = 4

print('Исходный список: ', myList)

myList.insert(3, myNum)

print('Список со вставленным элементом %d' % myNum, myList, end="\n\n")

# pop method

myList = [1, 2, 3, 4, 5]

print('Исходный список: ', myList)

myNum = myList.pop()

print('Список без последнего элемента %d: ' % myNum, myList, end="\n\n")

# remove method

myList = [1, 2, 3, 4, 5]

print('Исходный список: ', myList)

myList.remove(2)

print('Список без элемента с индексом 2: ', myList, end="\n\n")

# reverse method

myList = [1, 2, 3, 4, 5]

print('Перевернутый список %s: %s' % (myList, myList.reverse()), end="\n\n")

# sort method

myList = [3, 2, 5, 6, 9, 8, 7, 0, 4, 1]

print('Список до сортировки: ', myList)

myList.sort()

print('Список после прямой сортировки: ', myList)

myList.sort(reverse = True)

print('Список после обратной сортировки: ', myList, end="\n\n")

# count method

haystack = [1, 2, 3, 1, 2, 3, 1, 1, 1, 1, 2]

needle = 1

print('Число повторений элемента %d в списке %s: %d' % (needle, str(haystack), haystack.count(needle)), end="\n\n")

# index method

haystack = [1, 2, 3, 5, 1, 5, 2, 7, 8, 9, 0]

needle = 5

print(f'Индекс первого слева элемента со значением {needle} в списке {str(haystack)}: {haystack.index(needle)}', end="\n\n")

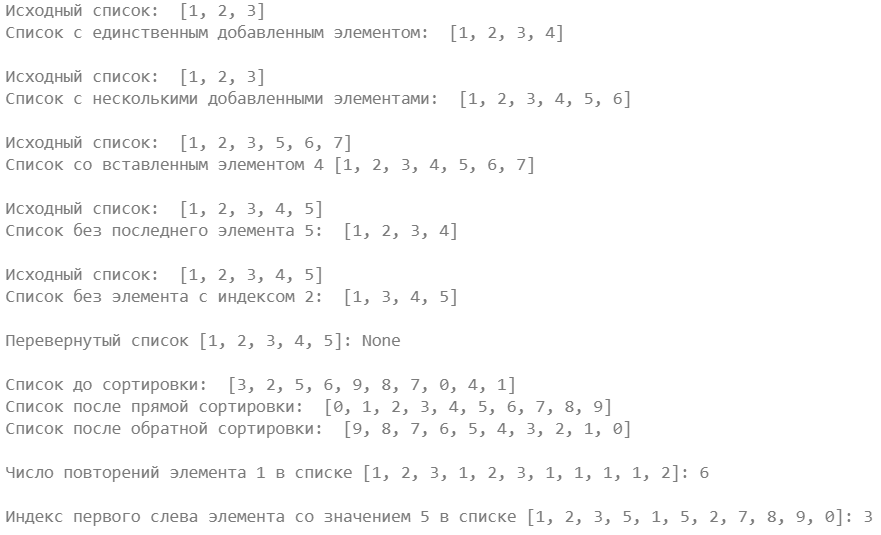


Рисунок ‎2.7 – Результаты работы методов обработки списков

## Выполнение задания 3

Продемонстрируем работу методов обработки словарей dict.clear(), dict.copy(), dict.fromitems(), dict.get(), dict.keys(), dict.values(), dict.items(), dict.pop(), dict.popitem(), dict.setdeafult(), dict.update() (рисунок ‎2.8) с помощью кода, приведенного в листинге ‎2.8.

Листинг ‎2.8 – Демонстрация методов обработки словарей

# clear method

fruit\_prices = { 'apple': 50, 'banana': 30, 'orange': 40, 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70, 'pineapple': 90, 'watermelon': 120 }

print('Оригинальный словарь: ', fruit\_prices)

fruit\_prices.clear()

print('Очищенный словарь: ', fruit\_prices)

print()

# copy method

fruit\_prices\_1 = { 'apple': 50, 'banana': 30, 'orange': 40, 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70, 'pineapple': 90, 'watermelon': 120 }

fruit\_prices\_2 = fruit\_prices\_1.copy()

key = 'apple'

fruit\_prices\_1[key] = 100

print('Цена фрукта %s в словаре 1: %d' % (key, fruit\_prices\_1[key]))

print('Цена фрукта %s в словаре 2: %d' % (key, fruit\_prices\_2[key]))

print()

# fromkeys method

fruits = ['apple', 'banana', 'orange', 'grape', 'lemon', 'pear', 'kiwi', 'mango', 'pineapple', 'watermelon']

fruit\_prices = dict.fromkeys(fruits, 100)

print('Цены фруктов: ', fruit\_prices)

print()

# get method

fruit\_prices = { 'apple': 50, 'banana': 30, 'orange': 40, 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70, 'pineapple': 90, 'watermelon': 120 }

# print(fruit\_prices['grapefruit']) # Приводит к исключению KeyError

print(fruit\_prices.get('grapefruit', 'Нет в продаже'))

print()

# keys method

fruit\_prices = { 'apple': 50, 'banana': 30, 'orange': 40, 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70, 'pineapple': 90, 'watermelon': 120 }

product = 'Фрукты: '

for key in fruit\_prices.keys():

product += f'{key}, '

product = product.rstrip(', ')

print(product)

print()

# values method

fruit\_prices = { 'apple': 50, 'banana': 30, 'orange': 40, 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70, 'pineapple': 90, 'watermelon': 120 }

product = 'Цены фруктов: '

for value in fruit\_prices.values():

product += f'{value}, '

product = product.rstrip(', ')

print(product)

print()

# items method

fruit\_prices = { 'apple': 50, 'banana': 30, 'orange': 40, 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70, 'pineapple': 90, 'watermelon': 120 }

product = 'Фрукты и их цены: '

for key, value in fruit\_prices.items():

product += f'{key}: {value}, '

product = product.rstrip(', ')

print(product)

print()

# pop method

fruit\_prices = { 'apple': 50, 'banana': 30, 'orange': 40 }

print('Словарь до удаления ключа: ', fruit\_prices)

fruit\_prices.pop('apple')

print('Словарь после удаления пары ключ-значение по ключу: ', fruit\_prices)

print()

# popitem method

fruit\_prices = { 'apple': 50, 'banana': 50, 'orange': 40 }

fruit\_prices['grape'] = 80

fruit\_prices['lemon'] = 25

print('Словарь до удаления последней добавленной пары ключ-значение: ', fruit\_prices)

fruit\_prices.popitem()

print('Словарь после удаления последней пары ключ-значение: ', fruit\_prices)

print()

# setdefault method

fruit\_prices = { 'apple': 50, 'banana': 50, 'orange': 40 }

print('Словарь до вызова setdefault() для несуществующего ключа: ', fruit\_prices)

print('Значение по умолчанию для нового ключа: ', fruit\_prices.setdefault('pear', 60))

print('Словарь после вызова setdefault(): ', fruit\_prices)

print()

# update method

fruit\_prices = { 'apple': 50, 'banana': 50, 'orange': 40 }

fruit\_prices\_1 = { 'grape': 80, 'lemon': 25, 'pear': 45, 'kiwi': 60, 'mango': 70 }

print('Оригинальный словарь до обновления: ', fruit\_prices)

fruit\_prices.update(fruit\_prices\_2)

print('Оригинальный словарь после обновления: ', fruit\_prices)

Изображение выглядит как текст, письмо, Шрифт, документ

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок ‎2.8 – Результаты работы методов обработки словарей

## Выполнение задания 4

Был написан код, позволяющий с помощью списковых включений получить из существующего списка новый список, содержащий только те строки исходного, длина которых превышает 5 символов, и которые записаны в нижнем регистре (листинг ‎2.9). Результат работы кода приведён на рисунке ‎2.9.

Листинг ‎2.9 – Код, демонстрирующий работу списковых включений

myList = [ "Hello", "world", "PYTHON", "programming", "TEST", "string", "AaBbCc", "lowercase", "UPPERCASE", "mixedCASE", "short", "verylongword", "123456", "abc", "ABCDEFGH" ]

resList = [x for x in myList if len(x) > 5 and x == x.lower()]

print(resList)

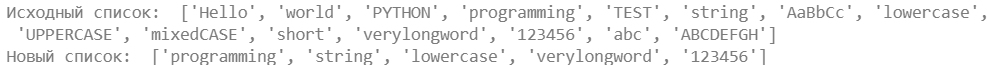


Рисунок ‎2.9 – Результаты работы кода со списковыми включениями

## Выполнение задания 5

Был написан и протестирован алгоритм быстрой сортировки (листинг ‎2.10). Результат работы алгоритма приведён на рисунке ‎2.10.

Листинг ‎2.10 – Команды для построения матрицы неточностей

import random

def quick\_sort(nums, low, high):

if low < high:

pi = partition(nums, low, high)

quick\_sort(nums, low, pi - 1)

quick\_sort(nums, pi + 1, high)

return nums

def partition(nums, low, high):

pi = low

nums[pi], nums[high] = nums[high], nums[pi]

pi = high

p = nums[pi]

i = low

for j in range(i, high):

if nums[j] < p:

nums[i], nums[j] = nums[j], nums[i]

i += 1

nums[pi], nums[i] = nums[i], nums[pi]

return i

if \_\_name\_\_ == '\_\_main\_\_':

arr = [random.randint(-100, 100) for \_ in range(10)]

print('Исходный список: ', arr)

print('Отсортированный список: ', quick\_sort(arr, 0, len(arr) - 1))



Рисунок ‎2.10 – Результаты работы алгоритма быстрой сортировки

## Выполнение задания 6.

Была решена задача 1 посредством изменения возвращаемого значения функции add() на a + b (листинг ‎2.11).

Листинг ‎2.11 – Доработанная функция add()

def add(a, b):

"Возвращает сумму a и b"

print('Passed a = %s and b = %s, returning a + b = %s' % (a, b, a + b))

return a + b

Была решена задача 2 посредством изменения функции buyLotsOfFruit() (листинг ‎2.12). Результаты тестирования функции приведены на рисунке ‎2.11.

Листинг ‎2.12 – Доработанная функция buyLotsOfFruit()

def buyLotsOfFruit(orderList):

    """

        orderList: Список-заказ из кортежей (fruit, numPounds)

    Возвращает стоимость заказа

    """

    totalCost = 0.0

    for fruit, weight in orderList:

        if fruitPrices.get(fruit) == None:

            print(f'Ошибка, фрукта {fruit} нет в списке!')

            return None

        totalCost += fruitPrices[fruit] \* weight

    return totalCost



Рисунок ‎2.11 – Результат тестирования функции buyLotsOfFruit()

Была решена задача 3 посредством изменения функции shopSmart() (листинг ‎2.13). Результаты тестирования функции приведены на рисунке ‎2.12.

Листинг ‎2.13 – Доработанная функция shopSmart()

def shopSmart(orderList, fruitShops):

    """

    Возвращает магазин с минимальной стоимостью заказа

        orderList: Список-заказ из кортежей (fruit, numPound)

        fruitShops: Список магазинов типа shop

    """

    minCost = float('inf')

    minFruitShop = None

    for fruitShop in fruitShops:

        cost = fruitShop.getPriceOfOrder(orderList)

        if cost < minCost:

            minCost = cost

            minFruitShop = fruitShop

    return minFruitShop

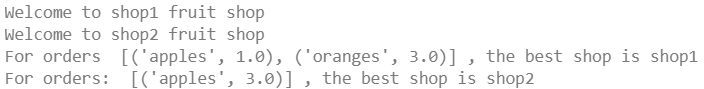


Рисунок ‎2.12 – Результат тестирования функции shopSmart()

С помощью скрипта autograder.py была оценена корректность работы разработанных функций (листинг ‎2.14).

Листинг ‎2.14 – Результат прохождения тестов

Question q1

===========

Passed a = 1 and b = 1, returning a + b = 2

\*\*\* PASS: test\_cases\q1\addition1.test

\*\*\* add(a,b) returns the sum of a and b

Passed a = 2 and b = 3, returning a + b = 5

\*\*\* PASS: test\_cases\q1\addition2.test

\*\*\* add(a,b) returns the sum of a and b

Passed a = 10 and b = -2.1, returning a + b = 7.9

\*\*\* PASS: test\_cases\q1\addition3.test

\*\*\* add(a,b) returns the sum of a and b

### Question q1: 1/1 ###

Question q2

===========

\*\*\* PASS: test\_cases\q2\food\_price1.test

\*\*\* buyLotsOfFruit correctly computes the cost of the order

\*\*\* PASS: test\_cases\q2\food\_price2.test

\*\*\* buyLotsOfFruit correctly computes the cost of the order

\*\*\* PASS: test\_cases\q2\food\_price3.test

\*\*\* buyLotsOfFruit correctly computes the cost of the order

### Question q2: 1/1 ###

Question q3

===========

Welcome to shop1 fruit shop

Welcome to shop2 fruit shop

\*\*\* PASS: test\_cases\q3\select\_shop1.test

\*\*\* shopSmart(order, shops) selects the cheapest shop

Welcome to shop1 fruit shop

Welcome to shop2 fruit shop

\*\*\* PASS: test\_cases\q3\select\_shop2.test

\*\*\* shopSmart(order, shops) selects the cheapest shop

Welcome to shop1 fruit shop

Welcome to shop2 fruit shop

Welcome to shop3 fruit shop

\*\*\* PASS: test\_cases\q3\select\_shop3.test

\*\*\* shopSmart(order, shops) selects the cheapest shop

### Question q3: 1/1 ###

Finished at 18:56:16

Provisional grades

==================

Question q1: 1/1

Question q2: 1/1

Question q3: 1/1

------------------

Total: 3/3

Your grades are NOT yet registered. To register your grades, make sure

to follow your instructor's guidelines to receive credit on your project.

Исходя из результатов работы скрипта, можно говорить об успешном прохождении разработанными функциями всех тестов.

# Вывод:

В результате выполнения лабораторной работы были изучены основы языка программирования Python. В частности, были рассмотрены основные арифметические и логические операции, 3 способа форматирования строк, а также различные структуры данных языка, а именно списки, кортежи, множества, словари. Были подробно изучены методы для работы со строками, списками и словарями. Кроме того, было получено базовое представление о срезах, списковых вложениях, функциях и лямба-выраженииях, или же анонимных функциях, а также об ООП, применяемом в Python.