**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Факультет Информационных технологий**

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА № \_**5**\_**

**Дисциплина: Backend-разработка**

**Тема:** Списки, кортежи, словари

**Выполнил(а): студент(ка) группы \_\_**221-379**\_\_**

\_\_\_\_\_Кодиров Жамшид Мурод угли\_\_\_\_\_\_

(Фамилия И.О.)

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_***

(Фамилия И.О., степень, звание) **(Оценка)**

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Замечания:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва**

**2025**

**Задание**

Цель работы: написать код, который демонстрирует работу со списками, кортежами и словарями в Python.

Требуется написать следующие конструкции:

Должны быть реализованы следующие конструкции:

1. Функция, принимающая на вход список.  
Функция возвращает перевёрнутый список.

2. Функция, принимающая на вход список.  
Функция изменяет одно, несколько или все значения списка.  
Функция возвращает изменённый список.

3. Функция, принимающая на вход два или более списков.  
Функция сравнивает переданные на вход списки.  
Функция возвращает отметку, равны или нет все переданные на вход списки.

4. Функция, принимающая на вход список и доп. параметры (необходимо самостоятельно их определить).   
Функция ДОЛЖНА иметь возможность выбрать диапазон значений из переданного списка с заданным шагом.  
Нужно рассмотреть все возможные ситуации, связанные с передаваемыми значениями.  
Функция возвращает список, соответствующий диапазону.

5. Функция, принимающая на вход некие параметры.  
Функция создаёт список, основываясь на переданных параметрах.  
Создание списка, его наполнение и возврат полученного списка.

6. Функция, принимающая принимающая на вход список и доп. параметры (необходимо самостоятельно их определить).  
Функция вставляет элемент в заданную позицию списка.  
Функция возвращает изменённый список.

7. Функция, принимающая на вход два или более списков и доп. параметры (необходимо самостоятельно их определить)..  
Функция объединяет все переданные на вход списки и сортирует их желаемым образом.  
Функция возвращает результирующий список.

8. Функция, не принимающая никаких параметров.  
Функция создаёт список из целых чисел произвольной длины.  
Функция проверяет, длина списка чётное число или нет.  
Если чётное, то функция сообщает об этом и создаёт новые списки до тех пор, пока не будет создан список нечётной длины.  
Если нечётное, то функция ищет центральный элемент списка и выводит количество элементов с таким же значением, что и у центрального элемента.

9. Функция, прибавляющая к первому списку другие списки.  
Если длина первого списка превышает заданный порог, необходимо удалить из списка некоторые элементы, чтобы число элементов списка не превышало порог.  
Функция возвращает изменённый первый список.

10. Минимум 6 функций, которые сортируют список по заданным критериям.  
Минимум 3 из этих функций ДОЛЖНЫ использовать функцию map()

11. Функция, которая извлекает с удалением минимальный элемент списка.  
Функция возвращает минимальный элемент списка.

12. Минимум 2 функции, принимающие на вход один или несколько кортежей, и, ВОЗМОЖНО, доп. параметры.  
Функции совершают некие операции над кортежем или кортежами.  
Функции МОГУТ возвращать некоторые значения.

13. Функция, принимающая на вход кортеж неопределённой длины, содержащий произвольные значения.  
Функция перебирает элементы кортежа и формирует новый кортеж, состоящий из типов данных каждого элемента входного кортежа.  
Функция возвращает кортеж из типов данных.

14. Функция, принимающая на вход кортеж и доп. параметры (необходимо самостоятельно их определить).  
Функция проверяет, есть ли заданный элемент в кортеже.  
Функция возвращает отметку, есть элемент или нет.

15. Функция, принимающая на вход один или несколько списков, и, возможно, доп. параметры.  
Функция формирует двумерный список по произвольным критериям и возвращает этот список.

16. Минимум 3 функции, которые принимают на вход словарь.  
Функции совершают некие операции над словарём.  
Функции возвращают какое-либо значениЕ, значениЯ.

17. Функция, принимающая на вход два или более словарей и доп. параметры (необходимо самостоятельно их определить).  
Функция считает, сколько раз встречается элемент с определённым ключом во всех словарях суммарно и выводит это значение (например, если есть 3 словаря, в двух из них есть элемент с ключом 'name', а в третьем нет, то выводится значение 2).

18. Функция, принимающая на вход комплексный словарь определённого формата, у которого будет минимум 3 уровня вложенности.  
Функция ищет в этом словаре определённый элемент или элементы, располагающиеся на самом последнем уровне вложенности.  
Функция возвращает значение найденного элемента или элементов или None, если такой элемент или элементы не найдены.

19. Функция, вызывающая все другие функции из шагов 1-18

**Дополнительные требования:**

Функции, созданные в шагах 1-18 ДОЛЖНЫ быть размещены в одном или нескольких отдельных файлах.

В файле main.py ДОЛЖНА быть конструкция if \_\_name\_\_ == "\_\_main\_\_", внутри которой ДОЛЖНА вызываться функция из шага 19.

В комментариях к каждой функции из шагов 1-18 ДОЛЖНО быть отмечено, к какому шагу относится эта функция.

КРАЙНЕ ЖЕЛАТЕЛЬНО, чтобы реализуемые функции имитировали какую-то реальную логику, были как-нибудь связаны между собой и содержали как можно меньше искусственных примеров.

ЗАПРЕЩАЕТСЯ копировать примеры из лекции.

РАЗРЕШАЕТСЯ использовать примеры из лекции за основу.

**Решение**

### 1: Функция reverse\_list без обработчиков исключений

**Описание:** Функция reverse\_list принимает список напитков и возвращает его в перевёрнутом порядке, используя срез с шагом -1.  
**Код в list\_functions.py:**

def reverse\_list(drinks\_list):

new\_list = drinks\_list[::-1]

return new\_list

**Вызов в main.py:**

print("1. Перевёрнутый список:", reverse\_list(drinks))

**Результат:** "1. Перевёрнутый список: ['Vodka', 'Wine', 'Beer']"

### 2: Функция change\_drinks без обработчиков исключений

**Описание:** Функция change\_drinks принимает список напитков и добавляет к каждому элементу строку " (cold)", изменяя исходный список.  
**Код в list\_functions.py:**

def change\_drinks(drinks\_list):

for i in range(len(drinks\_list)):

drinks\_list[i] = drinks\_list[i] + " cold"

return drinks\_list

**Вызов в main.py:**

print("2. Изменённый список:", change\_drinks(drinks.copy()))

**Результат:** "2. Изменённый список: ['Beer cold', 'Wine cold', 'Vodka cold']"

### 3: Функция compare\_lists без обработчиков исключений

**Описание:** Функция compare\_lists принимает три списка и проверяет, равны ли они все между собой, возвращая текстовую отметку.  
**Код в list\_functions.py:**

def compare\_lists(list1, list2, list3):

if list1 == list2 and list2 == list3:

return "Все списки одинаковые"

else:

return "Списки разные"

**Вызов в main.py:**

print("3. Сравнение списков:", compare\_lists(drinks, drinks.copy(), drinks.copy()))

**Результат:** "3. Сравнение списков: Все списки одинаковые"

### 4: Функция get\_range без обработчиков исключений

**Описание:** Функция get\_range принимает список, начальный индекс, конечный индекс и шаг, возвращая срез списка. Если параметры некорректны, возвращает сообщение об ошибке.  
**Код в list\_functions.py:**

def get\_range(drinks\_list, start, end, step):

if start < 0 or end > len(drinks\_list) or step <= 0:

return "Ошибка в параметрах"

new\_list = drinks\_list[start:end:step]

return new\_list

**Вызов в main.py:**

print("4. Диапазон:", get\_range(drinks, 0, 2, 1))

**Результат:** "4. Диапазон: ['Beer', 'Wine']"

### 5: Функция create\_drinks без обработчиков исключений

**Описание:** Функция create\_drinks создаёт список напитков заданной длины, добавляя к слову "Beer" номер.  
**Код в list\_functions.py:**

def create\_drinks(count):

drinks = []

for i in range(count):

drinks.append("Beer " + str(i + 1))

return drinks

**Вызов в main.py:**

print("5. Новый список:", create\_drinks(3))

**Результат:** "5. Новый список: ['Beer 1', 'Beer 2', 'Beer 3']"

### 6: Функция add\_drink без обработчиков исключений

**Описание:** Функция add\_drink вставляет новый напиток в заданную позицию списка, возвращая изменённый список. Если позиция некорректна, возвращает сообщение об ошибке.  
**Код в list\_functions.py:**

def add\_drink(drinks\_list, drink, position):

if position < 0 or position > len(drinks\_list):

return "Неправильная позиция"

drinks\_list.insert(position, drink)

return drinks\_list

**Вызов в main.py:**

print("6. Добавление:", add\_drink(drinks.copy(), "Rum", 1))

**Результат:** "6. Добавление: ['Beer', 'Rum', 'Wine', 'Vodka']"

### 7: Функция merge\_and\_sort без обработчиков исключений

**Описание:** Функция merge\_and\_sort объединяет три списка и сортирует их в прямом или обратном порядке в зависимости от параметра reverse.  
**Код в list\_functions.py:**

def merge\_and\_sort(list1, list2, list3, reverse=False):

big\_list = list1 + list2 + list3

if reverse:

big\_list.sort(reverse=True)

else:

big\_list.sort()

return big\_list

**Вызов в main.py:**

print("7. Объединение:", merge\_and\_sort(drinks, ["Rum"], ["Gin"], True))

**Результат:** "7. Объединение: ['Wine', 'Vodka', 'Rum', 'Gin', 'Beer']"

### 8: Функция check\_length без обработчиков исключений

**Описание:** Функция check\_length создаёт список случайной длины, проверяет чётность длины и либо урезает его до нечётной, либо считает повторения центрального элемента.  
**Код в list\_functions.py:**

def check\_length():

import random

length = random.randint(5, 10)

numbers = []

for i in range(length):

numbers.append(i)

if len(numbers) % 2 == 0:

print("Чётная длина, делаем нечётную")

while len(numbers) % 2 == 0:

numbers.pop()

else:

middle = len(numbers) // 2

middle\_value = numbers[middle]

count = 0

for num in numbers:

if num == middle\_value:

count += 1

print(f"Центральный элемент {middle\_value}, таких элементов: {count}")

return numbers

**Вызов в main.py:**

print("8. Проверка длины:", check\_length())

**Результат:** (пример) "Чётная длина, делаем нечётную"  
"8. Проверка длины: [0, 1, 2, 3, 4]"

### 9: Функция add\_lists без обработчиков исключений

**Описание:** Функция add\_lists добавляет элементы из двух списков в первый и урезает его, если длина превышает заданный порог.  
**Код в list\_functions.py:**

def add\_lists(main\_list, list1, list2, max\_length):

main\_list.extend(list1)

main\_list.extend(list2)

while len(main\_list) > max\_length:

main\_list.pop()

return main\_list

**Вызов в main.py:**

print("9. Добавление с порогом:", add\_lists(drinks.copy(), ["Rum"], ["Gin"], 4))

**Результат:** "9. Добавление с порогом: ['Beer', 'Wine', 'Vodka', 'Rum']"

### 10: Функция sort\_by\_length без обработчиков исключений (пример одной из 6)

**Описание:** Функция sort\_by\_length сортирует список напитков по длине их названий.  
**Код в sort\_functions.py:**

def sort\_by\_length(drinks\_list):

return sorted(drinks\_list, key=len)

**Вызов в main.py:**

print("10.1 Сортировка по длине:", sort\_by\_length(drinks))

**Результат:** "10.1 Сортировка по длине: ['Beer', 'Wine', 'Vodka']"

### 11: Функция get\_min без обработчиков исключений

**Описание:** Функция get\_min извлекает и удаляет минимальный элемент из списка, возвращая его. Если список пуст, возвращает сообщение.  
**Код в list\_functions.py:**

def get\_min(drinks\_list):

if not drinks\_list:

return "Список пустой"

min\_drink = min(drinks\_list)

drinks\_list.remove(min\_drink)

return min\_drink

**Вызов в main.py:**

temp\_list = drinks.copy()

print("11. Минимальный элемент:", get\_min(temp\_list))

**Результат:** "11. Минимальный элемент: Beer"

### 12: Функция tuple\_count без обработчиков исключений

**Описание:** Функция tuple\_count принимает два кортежа и возвращает сумму их длин.  
**Код в tuple\_functions.py:**

def tuple\_count(tuple1, tuple2):

total = len(tuple1) + len(tuple2)

return total

**Вызов в main.py:**

print("12. Сумма кортежей:", tuple\_count(bar\_tuple, bar\_tuple))

**Результат:** "12. Сумма кортежей: 6"

### 13: Функция get\_types без обработчиков исключений

**Описание:** Функция get\_types принимает кортеж и возвращает новый кортеж с типами данных его элементов.  
**Код в tuple\_functions.py:**

def get\_types(input\_tuple):

types\_tuple = tuple(type(x) for x in input\_tuple)

return types\_tuple

**Вызов в main.py:**

print("13. Типы данных:", get\_types(bar\_tuple))

**Результат:** "13. Типы данных: (<class 'str'>, <class 'int'>, <class 'str'>)"

### 14: Функция find\_in\_tuple без обработчиков исключений

**Описание:** Функция find\_in\_tuple проверяет, есть ли заданный элемент в кортеже, и возвращает текстовую отметку.  
**Код в tuple\_functions.py:**

def find\_in\_tuple(bar\_tuple, item):

if item in bar\_tuple:

return "Есть в кортеже"

return "Нет в кортеже"

**Вызов в main.py:**

print("14. Поиск в кортеже:", find\_in\_tuple(bar\_tuple, "Beer"))

**Результат:** "14. Поиск в кортеже: Есть в кортеже"

### 15: Функция make\_2d\_list без обработчиков исключений

**Описание:** Функция make\_2d\_list создаёт двумерный список из двух списков: напитков и их цен.  
**Код в list\_functions.py:**

def make\_2d\_list(drinks, prices):

bar\_menu = []

for i in range(len(drinks)):

bar\_menu.append([drinks[i], prices[i]])

return bar\_menu

**Вызов в main.py:**

print("15. 2D список:", make\_2d\_list(drinks, prices))

**Результат:** "15. 2D список: [['Beer', 5], ['Wine', 7], ['Vodka', 10]]"

### 16.1: Функция calculate\_inventory\_value без обработчиков исключений

**Описание:** Функция calculate\_inventory\_value принимает словарь с данными о напитках в баре (цена и количество) и возвращает общую стоимость всех запасов, перемножая цену на количество для каждого напитка и суммируя результат.  
**Код в dict\_functions.py:**

def calculate\_inventory\_value(bar\_dict):

total = 0

for drink in bar\_dict:

if "price" in bar\_dict[drink] and "quantity" in bar\_dict[drink]:

price = bar\_dict[drink]["price"]

quantity = bar\_dict[drink]["quantity"]

total = total + (price \* quantity)

return total

### 16.2: Функция get\_low\_stock\_drinks без обработчиков исключений

**Описание:** Функция get\_low\_stock\_drinks принимает словарь с данными о напитках и порог количества, возвращая список названий напитков, у которых запас меньше заданного порога.  
**Код в dict\_functions.py:**

def get\_low\_stock\_drinks(bar\_dict, threshold):

low\_drinks = []

for drink in bar\_dict:

if "quantity" in bar\_dict[drink]:

if bar\_dict[drink]["quantity"] < threshold:

low\_drinks.append(drink)

return low\_drinks

### 16.3: Функция get\_total\_stock без обработчиков исключений

**Описание:** Функция get\_total\_stock принимает словарь с данными о напитках и возвращает общее количество единиц товара на складе, суммируя значения по ключу "quantity" для каждого напитка.  
**Код в dict\_functions.py:**

def get\_total\_stock(bar\_dict):

total\_stock = 0

for drink in bar\_dict:

if "quantity" in bar\_dict[drink]:

total\_stock = total\_stock + bar\_dict[drink]["quantity"]

return total\_stock

**Вызов в main.py:**

    print("16.1 Общая стоимость запасов:", calculate\_inventory\_value(bar\_inventory))

    print("16.2 Напитки с низким запасом:", get\_low\_stock\_drinks(bar\_inventory, 5))

    print("16.3 Общее количество единиц:", get\_total\_stock(bar\_inventory))

### 17: Функция count\_key без обработчиков исключений

**Описание:** Функция count\_key принимает произвольное количество словарей (через \*dicts) и два параметра: ключ для поиска (key) и минимальное количество вхождений (min\_occurrences). Функция считает, сколько раз заданный ключ встречается в переданных словарях на верхнем уровне, и возвращает это значение. Если количество вхождений меньше min\_occurrences, возвращается сообщение об этом.  
**Код в dict\_functions.py:**

def count\_key (\*dicts, key, min\_occurrences=1):

total\_count = 0

for d in dicts:

if key in d:

total\_count += 1

if total\_count < min\_occurrences:

return f"Ключ '{key}' встречается {total\_count} раз, что меньше минимального значения {min\_occurrences}"

return total\_count

**Вызов в main.py:**

print("17. Подсчёт ключа (3 словаря):", count\_key (dict1, dict2, dict3, key="name"))

print("17. Подсчёт ключа (4 словаря):", count\_key (dict1, dict2, dict3, dict4, key="name"))

print("17. Подсчёт ключа (с minimum):", count\_key (dict1, dict2, dict3, key="name", min\_occurrences=3))

### 18: Функция find\_deepest без обработчиков исключений

**Описание:** Функция find\_deepest принимает на вход комплексный словарь с минимум тремя уровнями вложенности и ключ для поиска. Она рекурсивно обходит словарь, находя все значения, соответствующие указанному ключу, которые находятся на самом глубоком уровне вложенности. Если таких элементов нет, возвращается None. Функция также возвращает список всех найденных значений, если их несколько.

**Код в dict\_functions.py:**

def find\_deepest(bar\_dict, search\_key):

def recursive\_search(current\_dict, depth=0, max\_depth\_values=None):

if max\_depth\_values is None:

max\_depth\_values = {"depth": 0, "values": []}

*# Если это не словарь, прекращаем рекурсию*

if not isinstance(current\_dict, dict):

return max\_depth\_values

*# Проверяем текущий уровень*

for key, value in current\_dict.items():

if key == search\_key and not isinstance(value, dict):

*# Если нашли ключ и значение не словарь, обновляем максимальную глубину*

if depth >= max\_depth\_values["depth"]:

if depth > max\_depth\_values["depth"]:

max\_depth\_values["values"] = []

max\_depth\_values["depth"] = depth

max\_depth\_values["values"].append(value)

*# Если значение - словарь, продолжаем рекурсию*

elif isinstance(value, dict):

max\_depth\_values = recursive\_search(value, depth + 1, max\_depth\_values)

return max\_depth\_values

*# Запускаем рекурсивный поиск*

result = recursive\_search(bar\_dict)

*# Если ничего не нашли или глубина 0, возвращаем None*

if not result["values"] or result["depth"] < 3:

return None

*# Возвращаем список значений на максимальной глубине*

return result["values"]

**Вызов в main.py:**

print("18. Глубокий поиск (complex\_dict1, 'beer'):", find\_deepest(complex\_dict1, "beer"))))

**Результат:** "18. Глубокий поиск (complex\_dict1, 'beer'): ['Lager', 'IPA']"

### 19: Функция run\_all без обработчиков исключений

**Описание:** Функция run\_all вызывает все предыдущие функции с тестовыми данными и выводит результаты.  
**Код в main.py:**

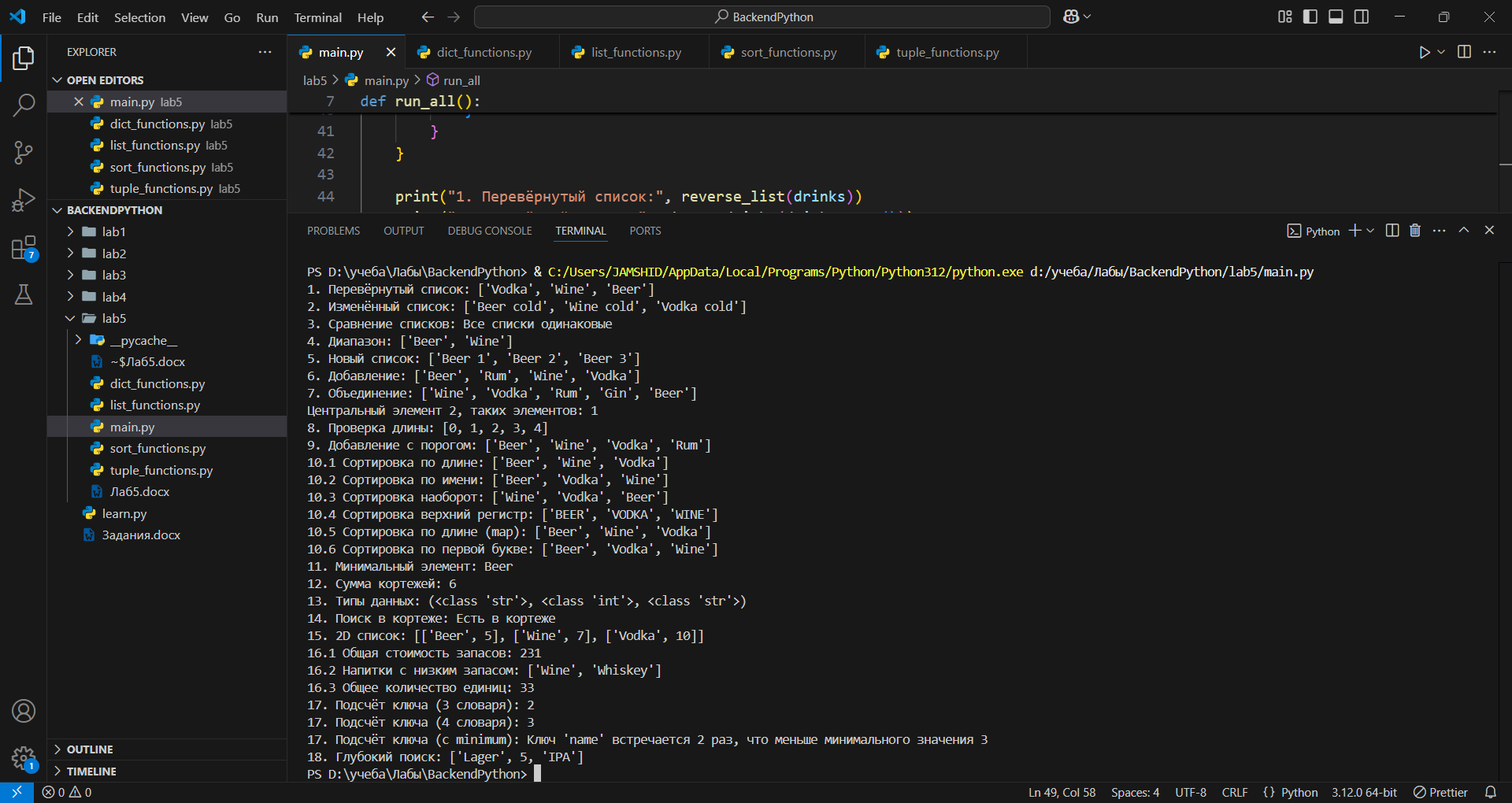
def run\_all():

**Вызов в main.py:**

if \_\_name\_\_ == "\_\_main\_\_":

run\_all()

**Результат:** Вывод всех результатов от 1 до 18.



Ссылка на Github: <https://github.com/QodirovJM/BackendPython>