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

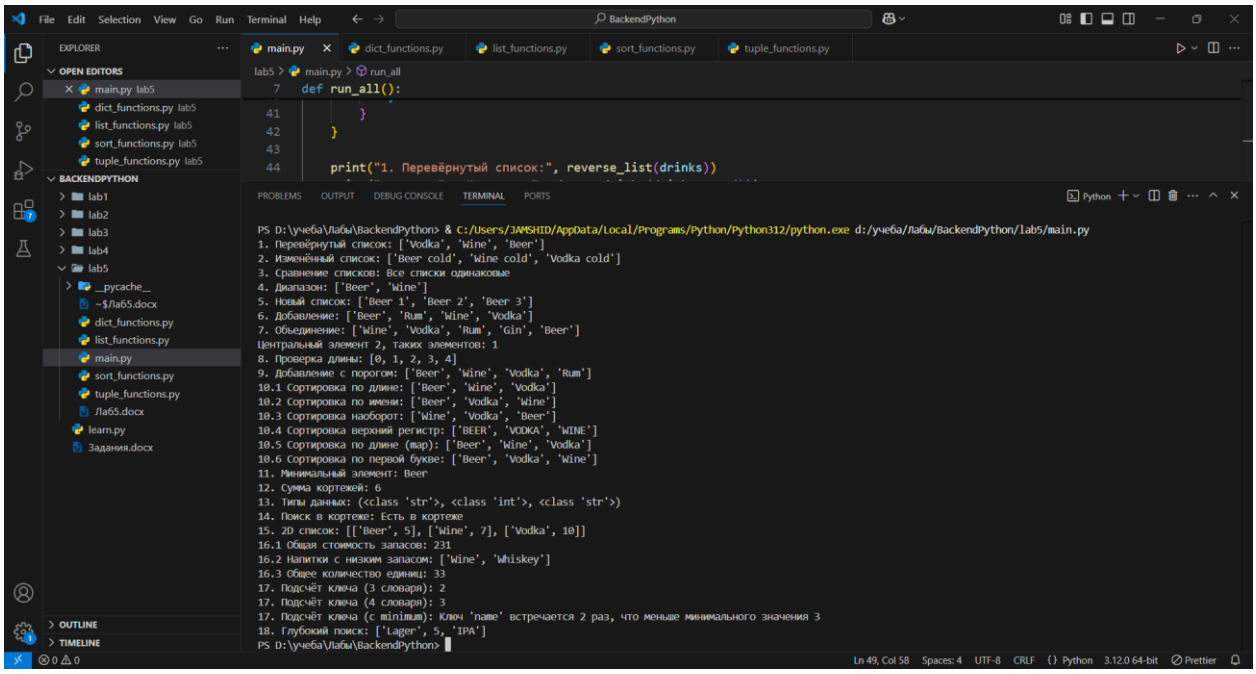
Описание: Функция run_all вызывает все предыдущие функции с тестовыми данными и выводит результаты.
Код в main.py:

```
def run_all():
```

Вызов в main.py:

```python
if __name__ == "__main__":
    run_all()
```

Результат: Вывод всех результатов от 1 до 18.



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